RADC-TR-88-324, Vol II (of nine), Part A
Interim Report
March 1989

NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT 1987 The Versatile Maintenance Expert System (VMES) Research Project

Syracuse University

S.C. Shapiro and S.N. Srihari

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-88-324, Vol II (of nine), Part A has been reviewed and is approved for publication.

APPROVED: Dale W. Richards

DALE W. RICHARDS
Project Engineer

APPROVED: John J. Bart

JOHN J. BART
Technical Director
Directorate of Reliability & Compatibility

FOR THE COMMANDER: John A. Ritz

JOHN A. RITZ
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (RBES) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific doucment require that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-88-324,Vol II (of nine), Part A
6a. NAME OF PERFORMING ORGANIZATION Northeast Artificial Intelligence Consortium (NAIC) 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (RBES)
6c. ADDRESS (City, State, and ZIP Code) 409 Link Hall Syracuse University Syracuse NY 13244-1240	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center 8b. OFFICE SYMBOL (If applicable) COES	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-85-C-0008
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 62702F; PROJECT NO. 5581; TASK NO 27; WORK UNIT ACCESSION NO. 13
11. TITLE (Include Security Classification) NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT 1987 The Versatile Maintenance Expert System (VMES) Research Project	
12. PERSONAL AUTHOR(S) S. C. Shapiro, S. N. Srihari	
13a. TYPE OF REPORT Interim 13b. TIME COVERED FROM Dec 86 TO Dec 87	14. DATE OF REPORT (Year, Month, Day) March 1989 15. PAGE COUNT 92
16. SUPPLEMENTARY NOTATION This effort was performed as a subcontract by the State University of New York - Buffalo to Syracuse University, Office of Sponsored Programs.	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Artificial Intelligence, Expert Systems, Fault Detection, Graphical Knowledge, User Interfaces, Reasoning, Semantic Networks
12	05		
12	07		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Northeast Artificial Intelligence Consortium (NAIC) is sponsored by the Air Force Systems command, Rome Air Development Center, and the Office of Scientific Research. Its purpose is to conduct pertinent research in artificial intelligence and to perform activities ancillary to this research. This report describes progress that has been made in the third year of the Consortium on the Versatile Maintenance Expert System research task. Technical areas addressed include device representation, modelling of connections, and interactions between logical and physical models. Investigation of analog components and differing levels of diagnosis was also initiated.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL DALE W. RICHARDS	22b. TELEPHONE (Include Area Code) (315) 330-3476 22c. OFFICE SYMBOL RADC/RBES

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

2A THE VERSATILE MAINTENANCE EXPERT SYSTEM (VMES) RESEARCH PROJECT

Report submitted by:

Dr. Stuart C. Shapiro, Principal Investigator
Dr. Sargur N. Srihari, Co-Principal Investigator
Dr. Mingruey R. Taie, Graduate Research Assistant
David B. Satnik, Graduate Research Assistant
Jiah-shing Chen, Graduate Research Assistant
Richard W. Wyatt, Graduate Research Assistant
James Geller, Graduate Research Assistant
Scott S. Campbell, Computer Programmer/Analyst

Department of Computer Science
SUNY at Buffalo
226 Bell Hall
Buffalo, NY 14260

Table of Contents

2A THE VERSATILE MAINTENANCE EXPERT SYSTEM (VMES) RESEARCH PROJECT

2A.1 TECHNICAL OVERVIEW

2A.1.1 Summary of 1987

Research on the Versatile Maintenance Expert System (VMES) project is concerned with issues in the development of a system that could diagnose faults in an electronic circuit and interact with a maintenance technician. During 1987 our research on this project had two distinct but overlapping phases: consolidation of work done during the previous two years and developing new directions of research.

In the first phase we emphasized several aspects of versatility. The aspects of "versatility" originally defined by the VMES project included: ability to diagnose a device that has been designed so recently that there has not been time to train technicians in its diagnosis; ability to diagnose a device that is so experimental that it has not been cost effective to design extensive automatic testing systems; ability to diagnose a wide range of devices from a similar family of devices; ability to serve at various maintenance levels (field, depot, etc.); ability to interact flexibly and in a "user-friendly" fashion. Significant achievements of the first phase of the VMES project, all of which were consolidated in the first half of 1987, were in the area of device representation and fault diagnosis. These were: development of an improved diagnostic strategy that uses suspect ordering criteria, introduction of a new model of connections that enables VMES to diagnose connection problems such as interrupted wires and bad contact points, and incorporation of both logical and physical device models to ease user interaction and speed up diagnosis.

In the second phase of our work we began to design an improved system which emphasizes the interactions between the logical and physical models. This was made possible by our acquisition of an actual working physical device. We also began to explore how to accommodate analog components, in addition to the digital components previously handled. A more complex control structure to handle diagnosis at different levels of knowledge began to take shape.

2A.2 PHASE 1: DEVICE REPRESENTATION AND FAULT DIAGNOSIS[1]

The major achievements of the VMES project in the first phase are in the areas of device representation and fault diagnosis. These were: the diagnostic strategy was improved by using more effective suspect ordering criteria, a new model of connections was introduced to enable VMES to diagnose connection problems such as interrupted wires and bad contact points, and both logical and physical device models were incorporated to ease user interaction and to speed up diagnosis. The remainderof this section expands upon these three topics.

2A.2.1 Suspect Ordering

Suspects are first sorted into sublists by *global criteria* called *fault possibilities*. Fault possibility is determined by evaluating the suspects against the overall current situation, which consists of the current test results. For the current implementation, there is only one global criterion: a suspect has higher fault possibility if it contributes to more vio-expct output ports. Suspects within each sublist are then sorted by some *local criteria* called *fault potentialities*. Fault potentiality is a measure of the rate a particular type of component may fail. It is independent of the environment, depending only on the component type. (It may also depend on the lot number of the component, but so far we do not treat such details.) The ideal fault potentiality data for our domain is the thermal analysis data of the components. Due to the unavailability of the thermal analysis data, it is now implemented as an index ranging from 1 to 3. Component types with no stored fault potentiality data default to 2.

VMES does not make the single fault assumption. The system incorporates the user's judgement by offering him an opportunity to terminate the diagnosis session whenever a faulty part is located. The user can choose to continue the investigation of the remaining suspects if he feels that more faults are possible or if he would merely like to make sure other suspects have no problems. The user may have another opportunity to terminate the diagnosis when VMES notices a diagnostic short-cut from the dual device model is present; this will be discussed in the section of "Dual Device Model".

2A.2.2 Modeling Connections

Model-based circuit diagnosis isolates the faulty components of a malfunctioning electronic device by reasoning on the basis of structural and functional description of the device. In this part of our work, we argue that explicit representation of wires and points of contact (POCONs) is necessary for diagnosing faults of circuit connections. The traditional model of a wire as a uni-directional module is inappropriate because it ignores the bi-directional nature of a wire and does not include POCONs. A new model of wires and POCONs and the corresponding semantic network representation are devised and implemented. A wire is modeled as a bi-directional module to preserve its physical property, and its uni-directional design intention is retained by the connection mechanism. A deliberate component connection mechanism - either by forming a POCON from two different ports or superimposing two same ports together - is also devised and implemented. In this new connection mechanism, components at different hierarchical levels are linked together by port superimposition, and components at the same hierarchical level are linked by POCON formation. The new wire model and connection mechanism are effective in circuit simulation and fault diagnosis. The resultant VMES has been successful in locating interrupted wires and bad contact points, and as has been discussed,

[1]This section is a summary of the following papers: Taie, M. R., Geller, J., Srihari, S. N., Shapiro, S. C., "Knowledge Based Modeling of Circuit Boards", in *Proc. of RAMS-87*, Jan. 1987; Taie, M. R. and Srihari, S. N., "Modeling Connections for Circuit Diagnosis", in *Proc. of the 3rd IEEE CAIA*, Feb. 1987; and Geller, J., Taie, M. R., Shapiro, S. C., Srihari, S. N., "Device representation and Graphics Interfaces of VMES", in *Proc. of the 2nd International Conf. on Applications of AI in Engineering*, Aug. 1987.

the new connection model makes it is possible to handle bridge problems to a limited extent.

2A.2.3 Dual Device Models

Human diagnosticians of electronic devices seem to simultaneously maintain models of the logical and physical structures of the target device. They carry out most of the diagnostic reasoning over the logical structure of the device due to its functional association. While carrying out the reasoning, the logical structure is apparently mapped to the physical structure from time to time. Tests and measurements are first initialized using the logical structure, and are then realized and executed on the physical structure. Repair, which is usually done by replacing a physical unit or by fixing a physical connection, is planned and done on the physical structure. In other words, maintenance technicians use a model of physical structure of the target device, which is a hierarchically arranged set of replaceable physical components at various maintenance levels such as field-level and depot-level. By mapping the logical structure of the device to its physical equivalent, maintenance technicians are able to terminate the diagnostic process at the right moment and to form an adequate repair plan.

Given that the mapping between the logical structure of the device and its physical equivalent happens throughout the diagnostic process at all hierarchical levels, the speed in carrying out the mapping is critical to the time needed to locate faults. This implies that objects on both the logical structure and the physical structure of the device should be closely linked to each other so that the mapping is done efficiently. Even experienced technicians may have difficulty in locating a point on a schematic diagram of the real device, where the schematic diagram represents the logical structure of the device, and the form of the real device is the physical structure. This difficulty is attributable to the large difference between the logical and the physical structures and a lack of cross-links at all hierarchical levels of the device in human memory. On the other hand, when modeling and representing a device in an automatic fault diagnosis system, the cross-links between its logical structure and physical structure can be modeled and represented to an appropriate level of detail. This is indeed possible to do in a computer with reasonably sized memory.

In VMES, the physical structure of a device is represented distinctly from but in a similar way as its logical structure. In a structural template for a logical component type, every subpart of the component type is specified with a subpart "id" and a subpart "type", which are used to instantiate the subpart if it is found to be a suspect and further investigation of it is necessary. In addition to the subpart "id" and "type", an "mntn-lv" indicator is also associated with every subpart of a physical component type. The "mntn-lv" indicator shows the intended maintenance level of the subpart, i.e., the maintenance level where the subpart, if found faulty, is replaced without further diagnosis. The "mntn-lv" indicator is associated with the physical structure rather than the logical structure of a device to reflect the fact that human experts form and carry out a repair plan based on a physical model rather than a logical model of the device.

In order to abstract a device into a model, which can be efficiently represented and interpreted, some abstraction restrictions have to be made. First, the hierarchical trees abstracted from the two perspectives should have the same number of hierarchical levels. Second, the cross-links can only be made at the same hierarchical level. Third, several logical objects on the logical structure can correspond to the same physical object on the physical tree, but a logical object cannot spread over several physical objects. This restriction seems unreasonable at first, but a closer investigation of the electronic domain shows the contrary — physical objects in the domain usually have larger grain size than logical objects. This is especially true with modern technology as more and more logical functioning units are being packed into a physical unit, e.g., a simple HexInverter chip (a physical object) contains six independent inverters (logical objects).

The two representations of the logical and the physical structures of a device are cross-linked at every hierarchical levels. There are two kinds of cross-links between the logical structure and the physical structure of a device. The first kind are cross-links for components. The second kind are cross-links for ports. While the cross-links of components helps in determining if the diagnostic process should go on or terminate, and in forming a repair plan, the cross-links of ports makes user interaction much easier — when ordering a test or a measurement, it can be used to clearly direct the user to the right location on the real device. The advantage of a logical abstraction of the device is that it provides a high level view of the device which facilitates the diagnostic reasoning. For instance, a n-bit wire is abstracted as a single logical wire, thus freeing the technician (or a fault diagnosis system) from thinking about bit slices. However, when a measurement is required, it is necessary to locate all the bit-ports on the real device, and this is often a difficult task for human diagnosticians since these bit-ports may spread out randomly.

In the rest of this section, we describe how VMES uses this device representation to facilitate fault diagnosis and user interaction. When bad outputs are found in the suspect currently being investigated, the system has to determine if the diagnosis should terminate or not. Most fault diagnosis systems use the simple idea of SRU (smallest replaceable unit) which says that the diagnostic process stops when the current suspect is a SRU, i.e., a terminal node (a leaf) of the structural hierarchical tree of the device VMES takes a more flexible approach by incorporating the idea of "intended maintenance level" into the system. A system parameter, VMES.IML, is set to the "intended maintenance level" the system is working on. If a part shows some bad outputs and it is at the intended maintenance level, it is declared faulty and the diagnosis on it is terminated. For example, a board is replaced at *field* and then sent back to a *depot* where the fault is further isolated to a chip. The checking for the maintenance level of a part is done on the corresponding physical object of the part (a logical object), and a repair plan is formed based on the component type of the physical object. VMES also provides an opportunity for the user to short-cut the diagnosis by noticing that all remaining (logical) suspects are in a single replaceable physical unit at VMES.IML. Since the same physical object gets replaced no matter which logical suspect is faulty, further discrimination among the suspects is unnecessary, provided that connections are assumed to be intact.

The major interaction between VMES and the user is the input of port values. Since diagnostic reasoning is carried out on the logical model of the device, VMES always wants the values of the logical ports. Through the cross-links between logical and the physical structures, VMES is able to inform the user which "physical port" should be measured to obtain the value for a given logical port. For representation and display efficiencies, wires are excluded from the physical representation of a device; this does not hurt the user interaction since the wire-end of a wire can always be identified as the wire-end connected to a port of a common component in the physical representation.

The third use of the physical representation of a device is in repair suggestions. When a faulty object is found or the end of the diagnosis session is reached, VMES suggests a repair plan to the user according to the type of the faulty object. If the faulty object is a common component, VMES just suggests that the user replace its corresponding physical part. If it is a wire, the corresponding physical wires are identified for repair. Note that a logical wire may correspond to several physical wires, for example, a 4-bit logical wire may be realized by four wires on a printed circuit board. Only the physical wires which are responsible for the fault are identified for repair. This is done by decomposing the port value of a logical wire into bit slices to determine which bit(s) are giving incorrect values. Finally, if the faulty object is a POCON, that is, it is a bad contact point, the user is directed to the location of the contact point. The physical representation is not only used to form the repair plan, it also helps direct the user to the object or location on the real device where the repair is actually performed. In other words, it provides for better user interaction in both test and repair.

2A.3 PHASE 2: A NEW DIAGNOSTIC ARCHITECTURE

This section explores the new areas and directions of the VMES project, and, by contrasting them with those of the existing system, exhibits the reasons for pursuing those areas and directions. The following details the progress of the current research assistants on the VMES project since August.

The first month (August) was devoted almost entirely to learning. A number of background papers on fault diagnosis were read and Mingruey Taie's recent dissertation was studied in depth. From these readings much was learnt about fault diagnosis in general and the existing implementation in particular. A working knowledge of SNePS, which is used extensively in the implementation, and of the updated version SNePS II, which is to be used in the future implementation, was also acquired.

As soon as there was sufficient familiarity with the existing implementation, a new device from the AOTS system was selected for implementation. A device was wanted which was sufficiently similar to the already implemented PCM6 board so as to be able to be implemented without modifying the existing system, but which would still afford an adequate a test of the system robustness. The valuable experience of implementing a new device was also a major consideration. The PCM4 board was selected. This board had three new types of digital and two new types of analogue component and thus would test the ability of the system to handle new components. The decision was partly influenced by the fact that there was a picture of the physical PCM4 board. Access of some kind to the nature of the physical device was important since there had already arisen the intention to make greater use of the distinction between the logical and the physical representations of a device employed by the existing system.

As preparation to implementing the PCM4 board, it was decided to implement a very small device. A half adder was chosen and was implemented quite quickly. The implementation of the PCM4 board itself has just recently been completed. These implementations provided much information about what the system could, and could not, do. From the knowledge acquired from the readings and the experience of the implementation, an overall plan for modifying, upgrading and extending the system evolved.

The new system to be built will begin with the knowledge representation used by Mingruey Taie and the control structure developed by Zhigang Xiang. It was deemed important to have continual access to an actual physical device while developing the new system so that greater account can be made of the physical representation. A physical device – a Heathkit Printer Buffer, Model SK-203 – was therefore purchased and assembled. Three general areas on which detailed work needs to be done have emerged:

(1) The refinement and adaptation of Xiang's general diagnostic architecture to the particular domain of circuit diagnosis. A control structure needs to be developed which will allow a more modular approach to the different kinds of circuit that are to be handled. If the more complex system to be built is to handle a greater variety of fault efficiently, a significantly more complex control structure is required. Such a structure must have the capacity to respond dynamically to the diagnostic process. Integrated into this structure will be a sub-structure controlling the repair strategy.

(2) The adaptation of the current means of knowledge representation developed by Taie to the new system. That representation is to be extended significantly so as to accommodate analogue components in general, and sequential rather than just combinational circuits.

(3) The improvement of the means of reasoning about the knowledge representation system in general, and especially about the means of generating, ordering and eliminating the fault suspects. It is hoped that use can be made, via the knowledge representations, of the natures of individual components, rather than of just the interconnections between them. Other methods of improvement must be explored also.

2A.3.1 Modularity of the Implementation

One of the important characteristics for the new implementation of VMES is modularity. We envision a system with several different types of reasoning techniques. For example, there may be specialized techniques for eliminating suspects in analog circuits that are not appropriate for combinational digital circuits. It would be desirable to be able to just use the portions of the system that are needed to diagnose the given device, rather than having the entire system in memory and only using a small portion. This is attractive because it is space efficient.

The existence of reasoning modules that can be requested by the user either when a diagnosis is started or even in the midst of a session is attractive from the experimental stand point as well. If specialized modules are designed, they can be tested with a "minimal" system configuration that will prevent other modules from interfering with the new module. If conflicts do arise between modules, the problems can be located by removing possible "suspect" modules until the problems are resolved and then focusing on the set of suspect modules that give rise to the observed problems.

The idea of a minimal system is that there are some set of modules that are necessary for the system to do any work. The modules in a minimal system would include input and output routines, routines to build the required device representations, the basic diagnostic reasoning modules, and a simple routine to guide the system toward a completed diagnosis.

Another key feature that modularity adds is expandability. Because a protocol must be established between the modules in the system, a clear and concise method of adding new modules should develop. When this occurs, you not only have an environment suitable to test new modules, but also an environment suited to developing new modules.

2A.3.2 Types of Circuits

Roughly, there are three types of electronic circuits -- analog, combinational and sequential circuits. Each type has its own characteristics; for example, combinational circuits have simple logical signals (0 & 1), analog circuits must consider current in addition to voltage, and sequential circuits have an extra time aspect. Hence, different types of knowledge and methods of diagnostic reasoning are needed for different types of circuits. The knowledge representation primitives and diagnostic rules the current system provides are good for combinational circuits but it is not clear how to use the current system for general circuits of the other two types. Since analog and sequential circuits exist in most electronic devices, it is desirable to extend the current system to handle them.

Analog circuits

In addition to voltage, current plays an important role in analog circuits. Their behavior is described by Ohm's law and Kirchhoff's current and voltage laws. In order to diagnose an analog circuit, all the above knowledge must be represented in such a way that the system can use it to analyze the circuit and derive helpful information.

The major difficulty in knowledge representation for analog circuits is representing signals. Unlike the digital domain, the analog domain has a richer set of signals and most of them are continuous. For example, we have to represent input and output waveforms and the system must be able to compare and manipulate them. All these are currently under investigation.

Sequential circuits

A sequential circuit is specified by a time sequence of external inputs, external outputs and internal states. The external outputs are functions of both the external inputs and the present state of the circuit. The next state of the circuit is also a function of its present state and external inputs. Current functional representation does not include the time dimension and

thus needs to be extended.

We are considering a hierarchic representation for time so that functions can be represented at different levels of time hierarchy. In other words, the behavior of a sequential circuit is described using different granularities of time. As a result, sequential circuits can be diagnosed at various levels of time hierarchy in a way similar to the use of structural hierarchy is used in combinational circuits currently.

2A.3.3 Suspect Handling

One of the key components of the VMES system is the way the fault suspects are handled. From the measured values of the various ports and the functions representing the components in question, those output ports which are vio-expects are found; i.e., those whose value violates what would be expected on the strength of the measured values and the component functions. Given the vio-expect output ports, there are are three main tasks to be accomplished; namely,

(a) The initial generation of the fault suspects, of the list of all those component parts that would, if they were faulty, account for the observed fault or faults,

(b) The ordering of those suspects according to the likelihood of their being responsible for the observed fault(s).

(c) The subsequent elimination of those suspects that in fact are not faulty, by the testing of the components in question.

The Current System

These three tasks are handled by the current VMES system in the following way:

(a) The suspects are generated by locating all those components which have an output (or bi-directional) port leading to a vio-expect output port. For example, if I1 and I2 are the sole input ports of the eventual output port O, if C1 - Cn are the components lying between either I1 and O or I2 and O, and if, given the values of I1 and I2 and the functions representing C1 - Cn, O is a vio-expect, then every component C1 - Cn becomes a suspect.

(b) The order of the suspects is determined by the global criterion that the more vio-expects a suspect contributes to, the more likely it is that it is faulty. (A local criterion that applies to suspects having the same global likelihood of being faulty has been proposed but has not been implemented.)

(c) Suspects are tested in the order of their likelihood of being faulty. The tests employ the existing values of the suspect's ports, and if these values are consistent with the suspect's function, it ceases to be a suspect.

The New System

Considerable improvements are envisaged for each of the three main tasks. Overall, more reasoning about the nature of the device, its components, and the connections between its components is desired.

One important kind of reasoning being explored is a qualitative reasoning that exploits a deep knowledge of the device. Knowledge of this general kind is already represented by the system. A human technician will ask himself whether the kind of fault he is observing is likely to be caused by a component of a given type, whether a failure of that component is likely to produce the observed fault. Here, he will make considerable use of his knowledge of how the device components function, and of the inter-connections between them. But he will not in general perform detailed or quantitative calculations about the various components at this stage of the diagnosis. Rather, he rely on comparisons and considerations of a more qualitative kind. (He may, of course, resort to detailed quantitative calculations at a later stage of the diagnosis.) It is in this sense that the reasoning is said to be qualitative.

The primary motivations for developing a reasoning system of the qualitative kind are these. First, since human technicians proceed in more or less this way, there is every reason to believe that the methods will yield quite a powerful diagnostic system. And second, the computational overhead of such a system is considerably less than that of a comparable system that relies on purely quantitative reasoning.

The new diagnostic system is dealing with the three main tasks as follows:

(a) Two methods of suspect generation are being examined. The first uses the kind of qualitative reasoning about the nature of the devices outlined above. In the second, the diagnostic system will request the user to measure the values at certain ports. Just which ports are measured is determined by the system. The measured ports are those that are logically intermediate between the input and output ports of which the values are already known by the system. In this way the system will home in on the trouble area more efficiently; large segments of the device can, at least initially, be excluded.

(b) The ordering of the suspects is particularly well suited to the qualitative reasoning approach. Other considerations will, of course, also be used. For example, the current method of ordering (mentioned earlier) will be retained, though it will now just be one of the factors that affects the suspect order. There is an interesting problem of how different ordering criteria are to be compared to yield the final suspect list.

(c) The kinds and numbers of tests to which a suspect is to be subjected is to be improved. The current system employs a single test whereas, in general, a number of tests is needed to eliminate a suspect. These tests should be sufficiently diverse to make it extremely likely that if the component passes, it is not faulty.

2A.3.4 New Test Device

Although this project is intended to develop a system that is adaptable to a wide range of devices in the domain of digital circuits without, at times, focusing on a specific device, this seems a hopeless task. By studying the problem of diagnosing faults in several specific devices and keeping the goal of versatility in mind, we should discover the constraints and methods needed to attack this problem more readily than if we were to study just one or two devices. Several devices have been modeled by the current implementation of VMES.

A new device was needed to help spur new ideas, extensions and refinements to the current theory. A primary concern was that the researchers' actually have a small piece of hardware to motivate the theory, rather than just schematics and pictures. After pursuing the possibility of using a piece of RADC equipment, it was suggested that we select an appropriate test device and purchase it.

The following criteria were used to select the device:

- reasonably priced (< $500.00)
- contains combinational circuitry
- contains sequential circuitry
- contains analog circuitry
- contains a variety of component types
- contains many logical components (>50)
- can be subdivided into several subdevices

The new test device that was selected was a Heathkit Printer Buffer Kit (model SK-203). In addition to satisfying the previously mentioned criteria, the device came with the circuit schematics, circuit board x-ray views that physically locate the components, an operation guide, the device specifications, some basic troubleshooting information, and block diagrams that are helpful in understanding the circuitry and components.

There is one other advantage worth mentioning about this device. Since we built the printer buffer from a kit we know more about its physical construction than we have about

any of the other test devices. Because of this and the fact that we actually have the physical device, we can cause faults in the device by removing or "damaging" components in a number of ways. Then we can use our implementation of VMES to diagnose the fault. Previously, testing an implementation of a device meant "damaging" a logical component in the representation and using VMES to locate that logical device. There was no physical device to check the diagnosis on. Now, we perform a more realistic test of our implementation by "damaging" a physical component (or more than one) and giving VMES the symptoms of the device. This gives us a much richer test device.

Physical Description

Model SK-203 printer buffer is enclosed in a sheet metal housing with the front and rear panels exposing the controls and fixtures. The enclosure itself is 9.25 inches wide, 1.5 inches high and 8 inches deep. It weighs approximately 1.5 pounds. On the front panel, as depicted in Figure 1, are 11 pushbutton switches (including for both input ports [1] a clear key, [2] a copy key, [3] a priority print key, [4] a restart key, and [5] an offline key, plus [6] a single swap key), three 7-segment LED displays and an LED above the "Swap" pushbutton. The arrangement of these parts are shown in Figure 1. FRONT-AND-REAR.

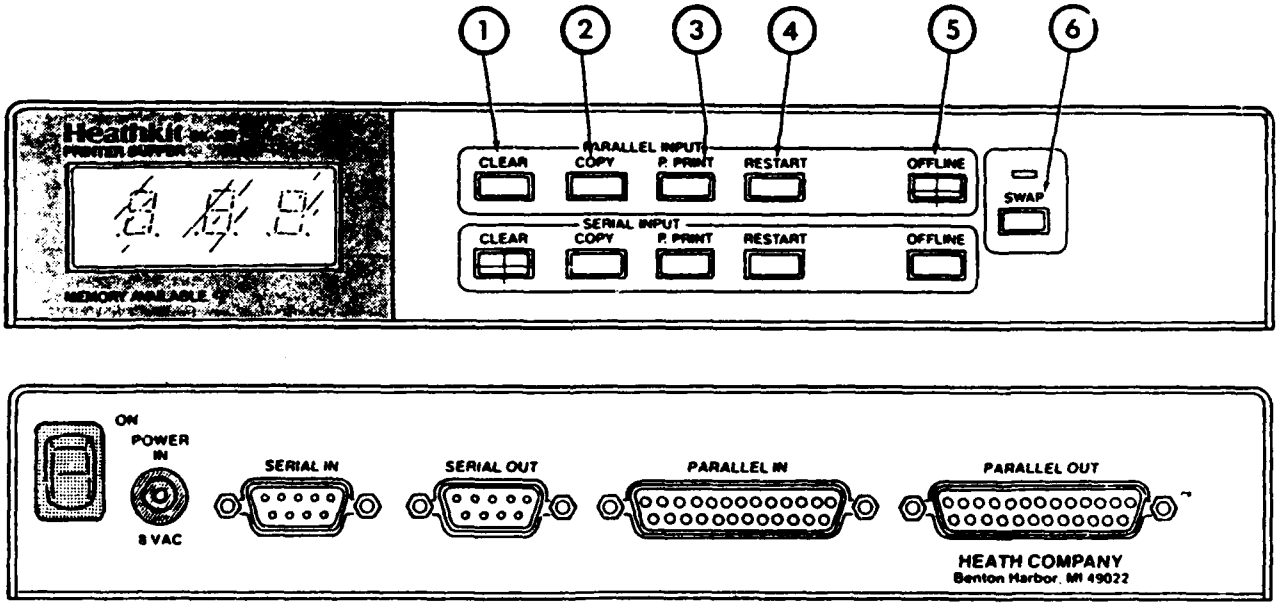


Figure 1. FRONT-AND-REAR

Also, there is an external transformer that plugs into a 120 VAC wall receptacle and outputs 8 VAC, 1 amphere to a plug that mates with the power jack on the rear panel.

Within the enclosure are two circuit boards connected by two 20-pin right angle plugs. The smaller circuit board mounts all of the components that are visible in the front panel of the enclosure. The main circuit board mounts the rest of the components of the printer buffer. This includes a 64180 CMOS microprocessor, eight 64K x 1 bit dynamic RAMs, an 8K ROM, and many logic and decoder IC's. In addition to these digital components there are various resistors, capacitors, diodes, and inductors that give us the analog components of the circuit.

Operational Description

The model SK-203 printer buffer will accept files from one or two computers and print them to one or two printers. By using a combination of the Swap key (pushbutton switch) and the Offline keys all the reasonable configurations can be achieved.

Some of the other features of the SK-203 printer buffer are:

- expandable memory up to 512 Kbytes
- ability to print up to 99 copies of a file
- allows the user to stop a current job to print another job and then resume the stopped print job
- the user can restart the printing of a file in case of printer failure
- both the parallel and serial ports are configurable by DIP switch settings

A complete description of the operation of the SK-203 printer buffer is available in the "Heathkit Manual for the PRINTER BUFFER Model SK-203", part number 595-3727-01.

2A.4 DEVELOPMENT OF SNePS-2

Most of this year has been spent on the development/upgrade of the Semantic Network Processing System (SNePS) for use with Texas Instruments Explorers and Symbolics Lisp Machines. The current complete version of SNePS, commonly referred to as SNePS79, is implemented in Franz Lisp, Opus 38.92 and currently running on Digital Equipment's VAX 750/780/785 family of computers. The new version of SNePS currently under development, commonly referred to as SNePS-2, has been chosen as the vehicle upon which the Versatile Maintenance Expert System (VMES) is to be implemented.

The development of SNePS-2 has been essentially broken down into two steps. The first step has been translating the SNePS code from Franz Lisp into Common Lisp — the language used on lisp machines. The second step is then the debugging of the new system in the completely new environment of the lisp machine.

2A.4.1 Motivation

In the summer of 1986, Ernesto Morgado[2] and the SNePS Research Group completed the core of a new version of SNePS, also implemented in Franz Lisp, Opus 38.92, which uses Morgado's theory of *Semantic Networks as Abstract Data Types* [Morgado86]. When the VMES project was put together, this new version of SNePS, referred to as SNePS-84, was selected as the base system.

When the Northeast Artificial Intelligence Consortium (NAIC) decided to purchase Texas Instruments Explorers for research and development, it was clear that this would be the target machine for our VMES project. One of the main problems with this was that the dialect of lisp used on these new machines was Common Lisp and not Franz Lisp as we have been using here at SUNY at Buffalo. Hence, the implementation of SNePS79 and SNePS-84, both written in Franz Lisp, Opus 38.92, would not run directly on these Explorers and some conversion of code would have to take place in order to use SNePS on this project.

Therefore, the project of converting the SNePS-84 system over to Common Lisp and the lisp machine environment was started. However, the new system created would not be just a direct translation of SNePS-84, it would be an enhanced version as well including a parser/generator for natural language and a graphical interface to the network.

2A.4.2 The Translation

During the past year, we have translated all of the code for SNePS-84 into the Common Lisp syntax. There are several differences between Franz Lisp and Common Lisp that make this a substantial task.

The easiest part of the translation is identifying functions which have different names in Common Lisp, and Franz Lisp. Occasionally, a function's arguments may need to be altered by the addition, deletion or re-arrangement of its parameters. This is usually due to additional flexibility exhibited by some Common Lisp functions to allow a user to specify how to manipulate the data. For example, the Common Lisp function *sort* allows the user to specify a predicate which will determine the ordering of the elements.

Another problem during translation is when functions available in one language are not available in the other. When functions avalable in Common Lisp were not available in Franz Lisp, the new must be written to take advantage of the efficiency of the new functions now available. In the worst case, it is clear that new code must be written.

---

[2] Morgado, E. J. M., "Semantic Networks as Abstract Data Types," Ph.D. Dissertation, Department of Computer Science, SUNY at Buffalo, September 1986.

Foremost in the problem of translation, is the rule of scope between the two languages. In Franz Lisp scoping occurs dynamically. This means that as a series of function calls are performed, calling environments, and related symbol bindings, *surround* subsequent function calls. Therefore, if a symbol does not have a value in the immediate environment, then the surrounding or calling environments are searched for the symbol. Lexical scoping used in Common Lisp, however, bases its search for symbol bindings based on the location in the code where that symbol is given a value.

2A.4.3 Enhancements

Since the project began, we have attempted to get SNePS-2 working on both TI Explorers and the Symbolics Lisp Machines in order to develop a system that was easily portable to either machine. Based on this desire, the availability of personnel, and their experience, some enhancements to SNePS have been implemented on one brand of lisp machine before the other.

The first enhancement is a graphical interface to the semantic network called *Ginseng*. This project was developed on the TI Explorer due to the availability of many primitive graphical functions found in the TI Graphics Toolkit — now included in version 3 of the Explorer's operating system. Such functions perform the actual task, for example, of drawing a line, circle or rectangle on the screen. Without these functions, it would have been necessary to write these functions ourselves to calculate the the shape of the desired items and draw them at the appropriate screen locations.

At this point in time, Ginseng is capable of displaying any SNePS node defined in a given network using the two functions *gi-desc* and *gi-dump*. Gi-desc is essentially a graphical version of the SNePSUL function *describe* which displays a tree consisting of the current node to be described and all nodes pointed to by descending relations from the current node. Gi-dump, which parallels the SNePSUL function *dump*, displays the current node and all nodes related by either an ascending *or* descending relation. These graphs, once displayed on the screen may then be printed by performing a screen dump to the department's QMS laser printer.

The second enhancement is the Natural Language INTerface (NLINT) which consists of a parser/generator based on an augmented transition network (ATN) grammar. The purpose of this is to read in sentences, based on a subset of English, and create the appropriate network to represent the inputted phrase. With the *surface*, and using the same grammar, one may ask for what a particular SNePS node represents in english and it will be displayed on the screen.

The NLINT system was initially translated from SNePS79, and hence Franz Lisp, onto the TI Explorers. At the time of the conversion, no particular test vehicle (i.e., grammar) was available to get all of the kinks out of the system. Subsequently, this feature was desired on the Symbolics machine and a great deal of time was taken to debug this system on that particular machine. As of yet NLINT has not been brought back to the Explorer, but I believe that this transition will be relatively effortless as the Explorers appear to be more relaxed in their Common Lisp syntax than is Symbolics.

2A.4.4 Future Direction

As stated, quite a bit of SNePS-2 is already converted to Common Lisp and operational. In fact, some additional packages to those previously mentioned, such as *Multi* (Multi-processing simulator) and *Match* (Network pattern matcher), are also working. The task that lies before us now is to integrate this into one large system, on each machine, and debug the SNePS Inference Package (SNIP). This is probably the most involved part of the conversion due to the fact that SNIP uses all of the other packages mentioned earlier.

Along with the debugging of SNIP is the introduction of the SNePS Belief Revisor (SNeBR) into the SNePS-2 system. Due to its integral part in how inference will be conducted, it is not clear as to whether the implementation of belief revision will occur concurrently with

or after the completion of SNIP. Perhaps we will defer work on SNeBR until after the completion of SNIP, while keeping in mind the kind of things that will be necessary when we do implement SNeBR.

2A.5 GRAPHICAL DEEP KNOWLEDGE: COGNITIVE BACKGROUND AND PROGRESS

Recent progress in the representation of spatial concepts like positions, whole-part relations, and reference frames is presented. The relations of graphical deep knowledge to cognitive science are shortly discussed.

The concept and importance of Graphical Deep Knowledge have been discussed in previous publications related to this project, *e.g.*, Geller, J., Shapiro, S. C., Graphical Deep Knowledge for Intelligent Machine Drafting, *Tenth International Joint Conference on Artificial Intelligence* Morgan Kaufmann Publishers Inc., Los Altos, CA, August 1987. In this paper we will review recent progress on the subject, as well as deepen the cognitive background of graphical deep knowledge research. Much of this material is derived from Geller, J., A Knowledge Representation Theory for Natural Language Graphics Dept. of Computer Science, State University of New York at Buffalo, (dissertation) 1987 (forthcoming)

2A.5.1 Recent Progress in Graphical Deep Knowledge Research

In Geller, J., Shapiro, S. C., Graphical Deep Knowledge for Intelligent Machine Drafting, *Tenth International Joint Conference on Artificial Intelligence* Morgan Kaufmann Publishers Inc., Los Altos, CA, August 1987, pp. 545-551, a number of different structures for position representation have been introduced. We have recently found that all the presented structures, and in fact a few more, can be summarized with one complex case-frame. This result does not make the individual structures unnecessary, because access to small simple structures is more practical from a pragmatic point of view. Nevertheless the summary representation by a single case-frame deepens our understanding of the concept of position. A similar overarching structure will also be shown for the three different types of part-hierarchies that have been introduced in Geller, J., Shapiro, S. C., Graphical Deep Knowledge *op. cit.* Finally a case-frame for representing simple assumptions about the currently valid reference frame has been found necessary and will be presented. We will start this section with a short reminder of the representational formalism used to explain case-frames.

2A.5.2 A Reminder: The Representational Format

According to the theory that is explained in Geller, J., A Knowledge Representation Theory for Natural Language Graphics *op. cit.*, we view the goal of our brand of AI research as achieving asymptotic domain coverage for knowledge in a specific topic area. There is a number of ways to specify such domain knowledge but in order to do concrete work one has to make a commitment as to which notational format to use, and as to what semantic theory to subscribe to. Our representational format is based on proposition nodes that dominate sets of arcs. While the proposition node is necessary to refer to a specific proposition, the information of the generic structure is completely contained in the combination of arcs. One can view this combination of arcs as a case-frame, and this is exactly what we will do. All structures shown are given as case-frames, while concrete examples are given in the linearized version of SNePS networks that was for the first time introduced in Geller, J., Shapiro, S. C., Graphical Deep Knowledge *op. cit.*

Concerning a semantic theory we will assume that the semantics of every case-frame is dependent solely on the combination of arcs, and will be specified by a descriptive semantics and a procedural semantics based on an interpreter for the structures. The semantics for one frame cannot be derived by combination of semantic primitives that are exemplified by arcs in the semantic network, but has to be specified individually. Globally speaking, the semantics of a whole network with many knowledge structures can be composed from the semantics of the single case-frames.

2A.5.3 Position Representations

Positions are the most complex phenomenon in graphical deep knowledge. Nevertheless it turns out that one can get by with a single structure to represent every possible case. A number of special cases can be derived from this structure that make practical applications easier to deal with. While the special cases have been discussed in previous reports, we will now present the syntax of the general purpose position case-frame.

Syntax:

```
object   <object>
relpos   x        <distance>
         y        <distance-2>
         z        <distance-3>
rel-to   <reference-object>
unittype          <unit-specifier>
space    <space-specifier>
```

All the terms written in angular brackets denote semantic network nodes, all other terms are semantic network arcs of the SNePS system. From a case-frame point of view, "object" represents a slot, while "<object>" stands for a slot-filler. The "relpos" slot contains a sub-frame which in turn has three slots x, y, and z.

Procedural Semantics:

In order to draw an object <object>, derive a displacement of <object> from the object <reference-object> by using <distance>, <distance-2>, and <distance-3> interpreted as measured in units of the unit-type according to <unit-specifier>, and create a pictorial projection according to the <view> expressed in an appropriate reference-frame assertion. If a coordinate transformation is necessary to fit the display on the screen, and if <space-specifier> is "world", perform the transformation. If the specifier is "screen", do not perform any transformation even if that results in failure of the display request.

Naturally, this procedural semantics is somewhat "hanging in the air", because no drawing will be possible unless a form for <object> is specified in the network. Objects might also have attributes, parts, etc. which will influence the final picture. All these representational constructs have been sufficiently covered in prior reports. The reference-frame assertion mentioned above is one of the newer results of this research and will be shown in a following section. Finally, the position of <reference-object> must be available in the network to correctly anchor the relative position expressions in the given case-frame. We will now give an example instantiation of this case-frame as a semantic network in a linear network description format.

Descriptive Semantics:

The descriptive semantics for the position case-frame will be given, based on the example shown before. This is done simply because it is easier to refer to a concrete structure than to an abstract case-frame. The semantic network node m9 represents the proposition that pcm-chip1 is 40 pixels to the right and 20 pixels below the position of trafo1 in the world. "World" hereby has to be understood as opposed to "screen", *i.e.*, the example deals with world coordinates that can be transformed to screen coordinates if necessitated by the given viewport on the screen and the actual values of x, y, and z.

The backslashes in front of the numbers transform the numbers into literal LISP atoms. This is necessary, because the numbers are treated as SNePS concept nodes, which requires the existence of a property list for them. Numerical atoms, however, do not have property lists.

Example:

```
m9(   object   pcm-chip1
      relpos   m8(   x    /40
                     y    /-20
                     z    /0
      rel-to   trafo1
      unittype       pixel
      space    world)
```

The sub-case-frame describing the actual coordinates is dominated by a molecular node (m8). While the node m9 has to be read as a proposition, m8 is read as a structured SNePS individual. m8 and all the structure below it represent an individual corresponding to a point in 3-dimensional space.

The above case-frame permits a number of variations that are not obvious from the given example, and that are part of the descriptive semantics. We will discuss them arc by arc.

(1) The x, y, z slots can contain fuzzy descriptors like "near" instead of numbers. In most practical cases the 2-d assumption is made, and the z slot is completely omitted. Under the standard reference frame, x and y are then equated with screen coordinate axes. The displacement expressed by x and y can also be inherited, in which case the relpos arc might be omitted.

(2) It is necessary to anchor position descriptions somehow, otherwise one gets an infinite regress of spatial references. For that purpose the rel-to arc may point to the special concept "world-center". Also the rel-to slot may be completely omitted if the reference object can be derived from the part hierarchy. This is the case if the given object is part of another object.

(3) The most common unit-type of measurement is "pixel"; however, the unit arc may also point either to the reference object at the end of the rel-to arc, or to the object itself, expressing so called body coordinates or reference object body coordinates which have been discussed in Geller, J., Shapiro, S. C., Graphical Deep Knowledge, *op. cit.*

(4) The space that a relative position specification refers to may be either "world" or "screen". If the space is "world", a display request will possibly result in a different screen position than world position. If the space is "screen", coordinates refer strictly to the screen and may not be transformed.

2A.5.4 Part Representations

As done with positions before, one can create an overarching structure to represent parts, such that all the part representations used in previous publications will be superseded by it. Nevertheless, to maintain a reasonable efficiency (in the AI sense of the word), the previously introduced special cases will not be discarded.

Procedural Semantics:

If the slot of <real-assem-clu> is taken by the concept "real-part", treat this case-frame as a real-part assertion. If <real-assem-clu> is "sub-assembly", treat it as an assembly assertion. If <real-ass-clu> is "sub-cluster" treat it as a sub-cluster proposition.

In previous publications the differences between these three types of part assertions have been explained. Real-parts are parts of an object that has its own drawable form such that it would be permissible to draw the super-part by itself, without its parts. Sub-assemblies are parts of an object that has its own drawable form, but drawing it without its parts is not permissible. Finally, clusters are abstract objects that have parts but no forms of their own. If

Syntax:

object	<object>
modality	<modality>
part-relation	<real-assem-clu>
sub-object	<object-2>

necessary, they may be represented by an empty box, for example.

In order to draw an object <object> it is necessary to retrieve form, position and possible attributes of the object as well as of its desired parts. The part hierarchy, in conjunction with user requests for simplified displays, will decide which parts actually to display. If a user wants to see a complete representation, all parts will be shown. If he wants a simplified representation, parts will not be shown, but sub-assemblies will be shown. If a complete display of a cluster is required, all the parts will be shown. If a simplified diplay of a cluster is required, the symbolic representation of the cluster as a box will be drawn, and all the sub-clusters will be omitted.

Descriptive Semantics:

Above structure asserts that <object-2> stands in the part-relation to <object> which is specified by <real-assem-clu>. For pragmatic reasons the three case frames presented in previous publications will be maintained.

2A.5.5 Reference Frame Representation

In the section on position representations a case-frame concerning the current reference frame was mentioned. This case-frame will be represented now. For NLG the problem of reference frame identification represents itself in the following manner. The screen that a person looks at naturally induces a coordinate system with axes parallel to the screen edges. There are a few reasonable choices for the center position, and we will assume the convention of having a horizontal x axes intersecting a vertical upward pointing y axes in the lower left corner of the screen. A person may then use screen coordinates to describe the location of an object. Unfortunately screen coordinates alone are an insufficient device for NLG.

A normal graphics device has a certain addressing range. Typically one has pixels from 0 to 1024 in the horizontal direction, and from 0 to 800 in the vertical direction. If a person refers to a position which is beyond this range, for instance by using negative coordinates, she makes it clear that she is not interested in screen coordinates, but in world coordinates. So reference frame identification means to determine the relation between the world coordinate system and the screen coordinate system.

It has been stressed before that this work deals only with 2-dimensional graphics. Nevertheless it turns out that people often look at a 2-dimensional diagram with a preconception that this is really a projection of a 3-dimensional world. Imagine a screen with nothing but a vertical arrangement of two circles on it. If one assumes that this scene represents a side-view, then one circle is clearly *above* the other circle. But if a person has just looked at a map, she might be biased to conceive of the diagram as a top-view. In this case the second circle is *behind* the first circle. In the side-view an object that is behind the first circle would be partly or totally occluded by it! So a natural language utterance can only be interpreted if one starts with the idea of a 3-dimensional world coordinate system. Luckily, people don't seem to assume arbitrary angles between world coordinates and screen coordinates, so it will be sufficient to deal with a few special cases.

What should be represented in the knowledge base of the system is the decison whether, if 3-d interpretation is necessary, diagrams should be considered side views or top views. Also the terms "left" and "right" are hopelessly ambiguous as used by people. We consider it important that the system should know explicitly and declaratively whether it is using these terms based on the user's view or its own view. This leads to the following knowledge structure.

Syntax:

view	<top-or-side>
leftness	<user-or-system>

Procedural Semantics:

If required to display an object A "behind" an object B, and the assumed view is "top" then A will be displayed vertically above B. If the view specified is "side" then A will be displayed at the same location as B, such that B is drawn later and overdraws A. "In-front", "above" and "below" are interpreted analogously. If required to display an object "at the left", and leftness is asserted as "user" ("system"), then the object will be displayed in the left ("right") area of the screen.

Example:

m1(	view	side
	leftness	user)

Descriptive Semantics:

The case-frame dominated by m1 asserts that any picture that needs to be interpreted as a projection of a 3-dimensional scene is assumed to be a projection orthogonal to the surface defined by the screen, assuming the screen is in its usual upright position. In addition it asserts that any reference to a lateral direction is assumed to be in the coordinate system defined by the body of the user. Without such an assertion the following questions would be meaningless to the system: "Do you mean left for me, or left for you?" and: "Is this a top view or a side view?". No other possible reference frames than "top" or "side" and "user" or "system" are permitted.

2A.5.6 A Look at Cognitive Science Contributions to GDK

We have recently extended our view of graphical deep knowledge, because it now is an integral part of Natural Language Graphics (NLG). It turns out that this permits us to investigate a number of interesting relations between GDK/NLG and cognitive science. Interactions with the three major fields of cognitive science, namely linguistics, cognitive psychology, and philosophy will be discussed. To our own surprise we have found that philosophy has contributed interesting ideas to our work, not only from the theoretical standpoint, but also from the practical view of user-interface design. The influence of linguistics and cognitive psychology goes more into the opposite direction. Interesting questions and hypotheses have been raised, to be answered by these two disciplines. Some pilot experiments have been performed to get psychological results.

2A.5.7 Yes, Philosophy can be Useful

The philosopher Paul Grice [Grice, H. P., Logic and Conversation, *Syntax and Semantics: Speech Acts*, 3, P. Cole and J. L. Morgan (eds.), Academic Press, New York, 1975, pp 41-59] has introduced the so called maxims of cooperative communication. These are pragmatic rules

about how to use natural language in communicating with other people. They are divided into maxims of quantity, maxims of quality, one maxim of relation, and maxims of manner. It turns out that these maxims can be applied to graphical communication with a GDK based NLG system. We will shortly point out some maxims which can be guiding for user interface design. (1) Make your contribution as informative as required. We have been able to develop this maxim into two sub-maxims that apply to NLG: (a) show all possible views of the same phenomenon, and (b) show a complete view of one phenomenon. (2) Do not make your contribution more informative than is required. One can interpret the use of part hierarchies for selecting objects in simplified displays as a conclusion of this maxim. (3) Do not say that for which you lack adequate evidence. This maxim can be used in displays that are combined with reasoning systems that are able to generate hypothesis. A graphical display of a hypothesis has to look different from a display of an established fact.

(4) Avoid obscurity of expression. This maxim can be interpreted as a requirement to use standard terminology whenever available. (5) Avoid ambiguity. In order to avoid ambiguity one has to avoid doubtful accidental features in graphical displays, like meaningless intersections. Also the number of icons used must be sufficient to cover all the phenomena of the given display domain. (6) Be orderly. This is of major importance. The order of drawing different parts in a graphics system is generally considered unimportant. All that counts is the final result. We disagree with this notion. A good NLG system should present the major features that a user is most interested in at the beginning of a new screen creation. In a maintenance system like VMES, it is important to first display faulty components, and then the rest of a device. This will permit a maintenance technician to react faster to machine output. For a deeper treatment of the subject, as well as for a long discussion of the maxim of relevance the user is again referred to Geller, J., A Knowledge Representation Theory, *op. cit.*

2A.5.8 A Short Look at Linguistics and Psychology

Practical work with the VMES user interface and its natural language component has led to questions that we have raised to linguists and psychologists. Among these questions were the following. What is the linguistic knowledge about the semantics of fuzzy spatial terms like "left" and "right"? Is there any experimental work that would help us determine what people think these terms mean? Are there cross linguistic studies on this subject? Most of our literature studies have resulted in negative answers. We have therefore conducted pilot experiments concerning the meaning of the spatial terms left, right, top, bottom, center, and the four corners. We have also constructed two hypotheses that limit the semantics of spatial terms, and that are open either to cross lingusitic examination or experimental work. Details of this work can be found in Geller, J., A Knowledge Representation Theory, *op. cit.*

2A.5.9 Conclusions

Three new case-frames have been presented that conceptually summarize a number of knowledge structures of graphical deep knowledge that have been published in previous reports. These are a general-purpose position case-frame, a new part case-frame, and a new case-frame for reference frame representation. It also has been pointed out that GDK has to be seen in the larger context of Natural Language Graphics, and that it derives interesting ideas from pragmatic rules of communication and raises interesting questions for natural language semanticists and experimental psychologists.

2A.7 VMES List of Publications

(1) Taie, M. R., Geller, J., Srihari, S. N., Shapiro, S. C., "Knowledge Based Modeling of Circuit Boards", in *Proc. of 1987 Annual Reliability and Maintainability Symposium*, IEEE, Philadelphia, PA, January 1987, 422-427.

(2) Taie, M. R. Srihari, S. N., "Modeling Connections for Circuit Diagnosis", in *Proc. of The 3rd IEEE Conference on AI Applications*, IEEE Computer Society Press, Orlando, FL, February 1987, 81-86.

(3) Srihari, S. N., "Applications of Expert Systems in Engineering", in *Engineering Progress of Western New York 6*, 2, Faculty of Engineering and Applied Sciences, SUNY at Buffalo, Winter-Spring 1987, 17-21.

(4) Taie, M. R., "Representation of Device Knowledge for Versatile Fault Diagnosis", Technical Report 87-07 (Ph.D. Dissertation), Department of Computer Science, SUNY at Buffalo, Buffalo, NY, May 1987.

(5) Geller, J., Taie, M. R., Shapiro, S. C., Srihari, S. N., "Device Representation and Graphics Interfaces of VMES", in *Knowledge Based Expert Systems for Engineering: Classification, Education and Control (a paper collection from the Second International Conference on Applications of AI in Engineering)*, D. Sriram, R.A. Adey (editors), Computational Mechanics Publications, Boston, MA, August 1987, 15-28.

(6) Geller, J., Shapiro, S. C., "Graphical Deep Knowledge for Intelligent Machine Drafting", in *Proc. of the 10th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, Milan, Italy, August 1987, 545-551.

(7) Shapiro, S. C., Geller, J., "Artificial Intelligence and Automated Design" *Computability of Design* Y. E. Kalay, ed., John Wiley & Sons, New York, NY, 1987, 173-187.

Appendix A1:

"Knowledge Based Modeling of Circuit Boards"

Taie, M.R.
Geller, J.
Srihari, S.N.
Shapiro, S.C.

Knowledge Based Modeling of Circuit Boards†

Mingruey R. Taie; SUNY at Buffalo; Buffalo

James Geller; SUNY at Buffalo; Buffalo

Sargur N. Srihari; SUNY at Buffalo; Buffalo

Stuart C. Shapiro; SUNY at Buffalo; Buffalo

Key Words: Maintenance, Fault Diagnosis, Expert System, Device Modeling, Device Representation, Graphical Knowledge, Circuit Board Analysis, User Interface, Knowledge Based Graphics, Knowledge Representation.

Abstract

This paper describes a maintenance expert system that has been designed with a focus on applied knowledge representation. Two main points of interest are described, the representation and reasoning mechanisms necessary for diagnosis based on a deep model of a device, and the representation for an integrated graphical user interface with limited natural language capabilities. Device structure is represented in a hierarchy of device types. Structural templates and instantiation rules permit focused diagnostic reasoning using lazy instantiation. Functional description is procedurally attached to the declarative network representation. Similarly, pieces of graphics code are attached to a declarative representation of the graphical appearance of the device.

Introduction

The VMES research project is aimed at the development of a versatile maintenance expert system for digital circuit troubleshooting. Theoretical and practical aspects of fault diagnosis, knowledge representation for system versatility, and knowledge-based graphical representation of target devices are being investigated.

VMES is designed to be versatile across a range of target devices in the chosen domain (a class of digital circuits); across most of the possible faults; across different maintenance levels; and across a variety of user interfaces. To achieve these versatilities, the device-model-based approach is followed. The device model-based approach, as opposed to the empirical-rule-based approach used by MYCIN [Shortliffe76a] for medical diagnosis and by CRIB [Hartley84a] for computer hardware fault diagnosis, is suggested to have advantages in knowledge acquisition, diagnosis capability, and system generalization [Davis83a, Davis84a, Genesereth84].

VMES is implemented in SNePS [Shapiro79a], the Semantic Network Processing System, and has several modules: an expandable component library as its knowledge base; an inference package (part of SNePS) with diagnostic rules; an active database for diagnosis; a user interface for intermediate users (engineers) to adapt VMES to new devices by incrementally updating the component library; and a multi-media user interface for end users (technicians) to interact with VMES for fault diagnosis. The architecture of VMES is shown in Figure 1.

Since a device-model-based fault diagnosis system reasons directly on the structure and function of a device and usually uses a simple inference engine, the representation of the device is vital to system performance. We use a hierarchical representation of knowledge to provide abstraction levels of devices. This allows a fault diagnosis system to focus on either individual objects or on several objects at a time.

The knowledge base stores descriptions of all component types used by the target devices. Objects which are parts of a device are instantiated only when needed. A formalism for device representation using instantiation rules and structural templates has been designed to describe the structure, the function, the intended maintenance level, and the test instruction of each component type to the knowledge base. The basis of the inference engine is SNIP, the SNePS Inference Package. It also includes an algorithm and diagnostic rules for carrying out the diagnosis. The active database is created for each diagnosis to store instantiated objects and their associated port values and states.

The multi-media user interface has menu, graphical, and limited natural language capabilities. Most human dialogues about a technical subject profit from the use of diagrams in addition to natural language communication. For circuits, technicians typically use wire plans which represent the function of a board, and structural diagrams which show the physical layout of the components of the circuit. The most natural way for a maintenance system to ask a user about the voltage value at a specific location is to display the structural diagram of the board and mark the position which is currently under focus, for instance by highlighting it. While CAD systems with similar interfaces exist, they are typically based on data objects like points, lines, etc. and do not store knowledge about visual properties of the *domain objects* that they are dealing with. In our approach all information necessary to display an object is stored as *knowledge* in a common framework with the knowledge necessary for doing diagnosis.

In this way we have combined our work on the graphical interface with more basic research in the representation of graphical knowledge. Systems of knowledge-based graphics have been reported in the literature by Zydbel et al [Zydbel81a] and by Friedell [Friedell84a]. By incorporating more and more knowledge into the graphical interface we have attained a new level of graphics that we want to refer to as *Intelligent Machine Drafting*.

Device Modeling

Compact representation is desirable for memory economy, diagnostic efficiency, and system versatility. In observing that many parts of an electronic device may have the same component type and thus show the same function, we find that representing every detail of a device will create unnecessary redundancy, which impairs system performance and versatility. Instead of representing all objects explicitly, VMES maintains an expandable component library, and objects are instantiated as needed. Devices are modeled hierarchically, and objects, which may be the device itself or its sub-parts at any hierarchical level, are represented as modules. Several implementations have been experimented with [Shapiro86a, Taie86a], and a formalism which represents devices by instantiation rules and structural templates is described here.

Structural Knowledge

The component library consists of descriptions of all component *types* used to construct the devices at all hierarchical levels. Each component *type* is in turn abstracted at two levels, and represented by a SNePS rule and a SNePS assertion. The former is categorized as an "instantiation rule", and the latter a "structural template".

At level-1 abstraction, knowledge about a component type is represented as an instantiation rule. The rule is used to instantiate an object of the component type as a module with I/O ports and associated functional description. The functional description is implemented as a LISP function that calculates the desired port value in terms of the values of other ports; this allows the simulation of the device.

At level-2 abstraction, a structural template is used to describe the sub-parts and wire connections of the object at the next hierarchical level. Component types and intended maintenance levels of sub-parts are also indicated. A structural template provides the necessary knowledge about the sub-structure of all objects of the same component *type* without representation overhead. Unlike instantiation rules, structural templates are never executed (fired) to produce a representation

† This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB DC 20332 under Contract No. F30602-85-C-0008.

for any specific object. When reasoning on the sub-structure of an object is required, instead of instantiating the sub-structure (all the sub-parts and wire connections) and then reasoning on the resulting representation, we do it directly on the structural template of the object, and only suspicious sub-parts are instantiated for further examination.

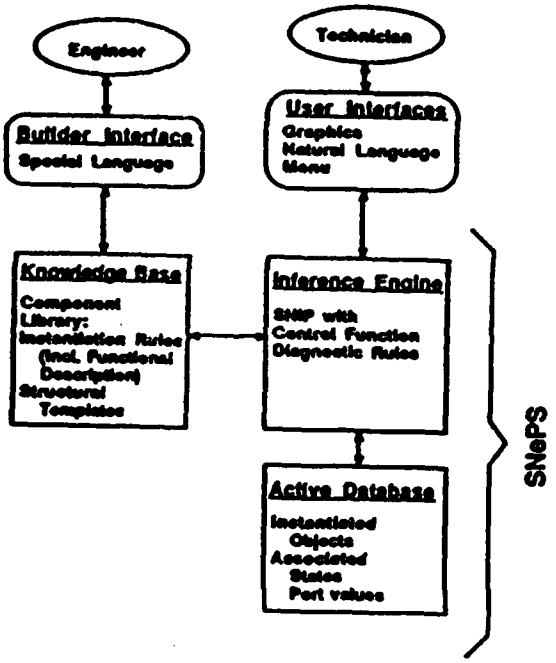


Figure 1. Architecture of VMES

All annotations are shown in italics.

```
(build avb $x
     ant (build object *x type PCM6 abs-lv IRFL1)
       The antecedent part states that for every object x if
       it is of type PCM6 and to be instantiated at level-1 by
       this rule then do the consequents. The first part of
       consequents builds the i/o ports of the object. The
       second part of consequents assigns the functional
       description to the output ports. The listing here is
       incomplete due to limited space. Below are some input
       ports for timing control.
     cq (build inport-of *x id t-shift) = vTS
     cq (build inport-of *x id r-shift) = vRS
       Below are some input ports for voice signal.
     cq (build inport-of *x id sigin1) = vSI1
     cq (build inport-of *x id sigin2) = vSI2
       Below are some output ports for voice signal.
     cq (build outport-of *x id sigout1) = vSO1
     cq (build outport-of *x id sigout2) = vSO2
       Below are some function assignments of voice signal
       output. It states that to calculate the port value of
       sigout1 (*vSO1), the function PCM6sigout with five
       arguments (pn 5) which are sigin1 (p1 *vSI1) and
       other four timing signals (p2-p5) is to be used, and
       the calculated result may have 5% of error tolerance
       (tolrnc 5).
     cq (build object *vSO1 func PCM6sigout tolrnc 5
            pn 5 p1 *vSI1 p2 *vTS p3 *vRS p4 *vT0 p5 *vR0)
     cq (build object *vSO6 func PCM6sigout tolrnc 5
            pn 5 p1 *vSI6 p2 *vTS p3 *vRS p4 *vT8 p5 *vR8)
```

Figure 2. Instantiation Rule for Level-1 Abstraction of Component *type* PCM6. (For testing purpose, the digital i/o pcmin & pcmout are connected for each channel to form a loop so that input at sigin is echoed to the output sigout.)

This model is illustrated in Figures 2 to 5, where figures 2 and 3 are annotated SNePS codes which build the SNePS representation for the two level abstractions of a component type PCM6, and Figure 4 and 5 are their pictorial equivalents. PCM6 stands for the six channel pulse coded modulation boards which are used for telecommunication.

There are three sections of a structural templates: the first one identifies that the template is for a particular type such as PCM6; the second section described the subparts; and the last section envisions the wire connections.

```
(build
   Section 1: Structural Template Identification.
 type PCM6 abs-lv STTL2
   Section 2: Subparts Description (incomplete). The id
   part gives the unique id of a subpart within the template.
   The ext-name part is used to extend the name of the
   subpart when it is instantiated. The type part gives
   the type of the subpart, and the maint-lv part indicates
   its intended maintenance level.
 sub-parts
    ((build id PCM6-pc1 ext-name PC1 type PCC
          maint-lv DEPOT)
     (build id PCM6-pc2 ext-name PC2 type PCC
          maint-lv DEPOT)
     (build id PCM6-nt0 ext-name NOTG.t0 type NOTG
          maint-lv DEPOT)
     (build id PCM6-nt1 ext-name NOTG.t1 type NOTG
          maint-lv DEPOT)
     (build id PCM6-ix1 ext-name IX1 type XFORM
          maint-lv DEPOT)
     (build id PCM6-ix2 ext-name IX2 type XFORM
          maint-lv DEPOT))
   Section 3: Wire Connections. (incomplete)
 connections (
       voice signal side ...
     (build from (build inport-of PCM6    id sigin1)
          to  (build inport-of PCM6-ix1 id in))
     (build from (build outport-of PCM6-ix1 id out)
          to  (build inport-of PCM6-pc1 id sigin))
       pcm i/o side ...
     (build from (build inport-of PCM6    id pcmin-A)
          to  ((build inport-of PCM6-pc1 id pcmin)
               (build inport-of PCM6-pc2 id pcmin)
               (build inport-of PCM6-pc5 id pcmin)))
       timing control ...
     (build from (build inport-of PCM6    id t0)
          to  (build inport-of PCM6-nt0 id in))
     (build from (build outport-of PCM6-nt0 id out)
          to ((build inport-of PCM6-pc1 id t-strobe)
              (build inport-of PCM6-pc3 id t-strobe]
```

Figure 3. Structural Templates for Level-2 Abstraction for Component *type* PCM6.

Functional Knowledge

Functional knowledge of a component *type* is represented as a procedural attachment to the semantic network. The functional description is usable to simulate the component behavior, i.e., to calculate the

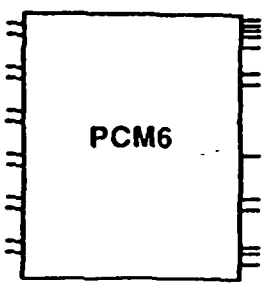


Figure 4. Level-1 abstraction of component type PCM6.

values of output ports if the values of the input ports are given. It should also be usable to infer the values of the input ports in terms of the values of other I/O ports. This is important if hypothetical reasoning is used for fault diagnosis. Though we have only used the functional description to calculate the value at the output port, our representation scheme can be used both ways.

The functional description is implemented as a LISP function, which calculates the desired port value in terms of the values of other ports. Every port of a component type has such a function associated with it, the link between the port and the function is described in the instantiation rule of the component type. Since different ports of different component types might display the same behavior, some functions can be shared. Figure 6 shows some examples of functional description.

Below is the function for the output port of ADDER-type objects

```
(defun ADDERout (inp1 inp2)
        (+ inp1 inp2))
```

Below is an example to show a function shared by several different component types namely by the type "PCM6", the type "wire" and the type "1-to-1 transformer". All these component types show the same behavior at our level of component abstraction: they echo the input to the output.

```
(defun ECHO (inp1)
        inp1)
```

Below is the function for the output port of ANDGate-type objects. (Other coding is possible, but since we use 1/0 for high/low, the following way is a convenient one.)

```
(defun ANDGout (inp1 inp2)
        (/ (+ inp1 inp2) 2))
```

Figure 6. Examples of Functional Description.

Graphical Knowledge

While the process of diagnosis is running, the user is informed about the activities of the system via a graphical trace of its reasoning.

The device is displayed on a graphics terminal, and parts currently under consideration are highlighted, for instance, by changing their color. Some more details about this will follow in a later section, here only representational questions will be raised.

In the chapters above we have introduced a notation for the knowledge used for diagnosis, which is based on the SNePS user language. All the information necessary to create a graphical representation of the diagnostic object is stored in the very same knowledge representation environment. This not only means that we are using the same SNePSUL syntax to describe objects in a way that pictures can be created, but we are using a common knowledge base, and in fact to a certain degree the *same knowledge* for the diagnosis and the drawing programs.

In this representation only primitive shapes are stored in a form comparable to "classical" graphics programs. For instance, the form of a multiplier is stored as a piece of code that, if executed, draws a multiplier. However any more complicated entity is stored declaratively in the network representations.

It is necessary to distinguish between two different types of graphical representations which pose different requirements. In a structural or physical representation a device is shown in a geometrically analog way. If a resistor is below a chip on a board, then one can expect the picture of a resistor below the picture of a chip on the corresponding plan. In order to construct such a plan a human as well as a system must have positional knowledge. This knowledge is usually expressed by coordinate pairs.

Functional or logical representations, on the other hand, do not need positional knowledge. If one draws a wire plan, it is not necessary to know exactly where to put a component. Certain connectivity conditions have to be satisfied in order to create a picture true to the object, and certain conventions of the draftsman's trade have to be observed, but there are no a priori rules that specify that a certain resistor must be under a certain chip. In fact not even neighborhood relations have to be preserved.

We will first describe the representation used for structural descriptions, and then talk about logical descriptions (wire plans). All the routines for structural display have been implemented, and the implementation for functional display is currently well on its way.

Graphics Structural Descriptions

It is necessary to know about the *form* of every object involved in the production of a drawing. In our system, forms are either linked directly to the corresponding object or an object inherits a form from a class of objects. This requires two case frames, one linking the object to a class and a second one linking the class to a form. Forms represent the link between the declarative and the procedural plane of the representation system. A form is at the same time two different things: it is a (base) node in the semantic network, and in this way accessible by the knowledge base handler, but it is also the name of a LISP function that contains calls to routines of a LISP graphics package. SNePSUL expressions for some of the caseframes described in this and the next section are given in Figure 7.

Positions are represented in a variety of different ways, the main ones being absolute positions, relative positions and relative sub-part positions. The caseframe for an object which is located at an absolute position contains an arc (a slot) to the object, and arcs to nodes denoting coordinate numbers. In the case of a relative position an additional arc identifies a "reference object". In order to draw a relatively placed object, the drawing program has to retrieve the position of the reference object first.

Relatively placed sub-parts do not have an arc to a reference object. In this case, the assumption made by the drawing system is that this part must be placed relative to its "super-object". In general the position of any part is assumed to be the position of its reference point. By convention the upper left corner is made the reference point whenever possible. So the caseframes described above relate to the reference points of the involved objects.

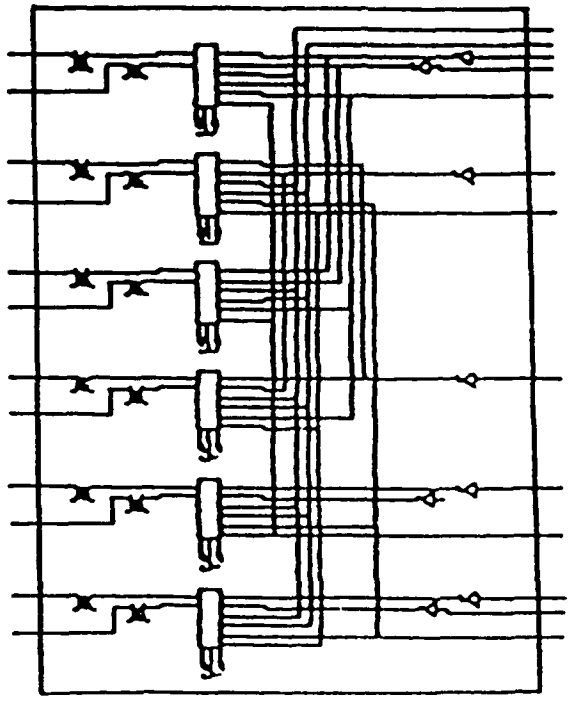


Figure 5. Level-2 abstraction of component type PCM6.

```
(build object D1-M1
      form    xmult
      modality function)
```
This describes an object with Individual Form

```
(build object D1-M2
      type    multiplier
      modality function)
```
This asserts that D1-M2 is a multiplier

```
(build class multiplier
      form    xmult
      modality function)
```
This expression links the class multiplier to the form xmult

```
(build object D1
      sub-parts (D1-M1 D1-M2)
      sub-assems (build inport-of D1
                        id      inp1)
      abspos (build x 100 y 200)
      modality function)
```
This is a partial description of D1. It has 2 parts D1-M1 and D1-M2, one sub-assembly which is an input port with the id inp1 and an absolute position at 100/200

Figure 7. Example Caseframes for Graphical Knowledge.

Our system also permits us to assign *attributes* to objects. We are discriminating between iconic and symbolic attributes. Iconic attributes are directly displayable, for instance "color" is such an attribute. Symbolic attributes require a mapping function that assigns a displayable attribute to a symbolic attribute. The diagnosis program makes heavy use of this possibility. For instance, the state (symbolic) attribute is displayed by mapping it to the color (iconic) attribute.

We are viewing attributes as functionals that take a form *function* and an attribute-value as arguments and return a modified form function. The mapping between symbolic attribute values and iconic attribute values is therefore done procedurally. For the example above the state "faulty" would be mapped to the color "red", and the state "suspect" would be mapped to "green".

Besides the class hierarchy, a *part hierarchy* is also employed using a caseframe with an object arc and one or more sub-parts arcs. It is possible for the user to specify how many levels of the part hierarchy he wants to see displayed.

Graphics Functional Representations

If one tries to create logical (functional) representations from a knowledge base, then the following interesting points become notable:

- *All the positional information can be eliminated.* It should not be necessary to specify any location.
- In order to create a reasonable picture, more knowledge of other types is necessary. For example, it is necessary to know which objects are inports and which objects are outports, because people usually expect the signal flow on a diagram to go from left to right, or from top to bottom. Incidentally, this information is also necessary for diagnosis, and therefore does not create any additional requirements.
- Forms loose their absolute meaning. Objects like wires and boards especially don't need to be specified by graphics code. In fact the forms of all wires will be the result of a routing algorithm. It is a well known discriminating factor between declarative and procedural representations that the latter do not permit incomplete specifications, while the former ones do. But this is exactly what one would like to have. All wires consist of vertical and horizontal lines, but nothing about the specific form of a wire is known before the positions of the impinging components have been fixed.

We also have found that, in general, for either type of representation:

- the classical part hierarchies are not sufficient even for most primitive applications. We have so far discriminated between real parts and sub-assemblies, and one more type of whole-part hierarchy for clusters might become necessary. Sub-assemblies are represented very similarly to sub-parts, only the arc sub-parts is replaced by the arc sub-assems.

The AI literature has so far discussed inheritance along the lines of a class hierarchy. We have *found it necessary to do inheritance along our part hierarchy*, and to control this inheritance with a meta-attribute. In this way a user can state declaratively whether he wants an attribute inherited or not. While we *do* want to inherit that a big chip has big ports we do *not* want to inherit that if a board is faulty, then so are all its parts!

Using no spatial information about the location of objects forces us to deal with placing and routing algorithms, however our objectives are far from a VLSI designer, and more comparable to the TYGES project [Eades86a] in that we are trying to create a graphically pleasing representation. We refer to this type of drawing activity as *Intelligent Machine Drafting*, a term that we have not yet seen in the literature.

We have defined a very limited class of circuits and are working on automatic placing of members of this class. The class has been defined formally, but in this paper we will limit ourselves to an intuitive summary description. Devices in the class consist of 0 or 1 main object and of parts of this main objects. Every part, as well as the main object, has "ports", and ports are connected by wires. The number of signal paths and the length of signal paths is expected to be small enough to permit placing them with our *column-equal-spacing* algorithm (which will be reviewed later). All components have to be connected "straight forwardly", *i.e.* no feedback loops are permitted. Icons (= parts) are all about the same size.

The column-equal-placing algorithm classifies elements according to their signal distance from the system input ports and assigns every part to a column. It then equally distributes all columns over the screen, *and equally distributes all parts inside their respective columns.* We are currently reviewing the literature on VLSI routing, in order to decide upon an appropriate routing algorithm.

The advantage of a system that does its own placing based on information about the structure of the device is obvious – to a large extent, creating a knowledge base *for maintenance purposes* takes care of creation of the necessary graphics. We have also found knowledge based programming to be far more robust than all other common programming paradigms. The reasoning system and the graphical interface of VMES were designed by different people with a minimum amount of personal interaction, nevertheless system integration did not pose any difficulties.

Another significant advantage of our representation system for graphical knowledge is the large amount of SNePS natural language processing software that can immediately be used. We have used the SNePS ATN package [Shapiro82a] to create a natural language interface to the graphics routines. This permits us to do *Natural Language Graphics* [Brown81a, Hussman84a] almost as a byproduct, the only step necessary was the creation of an ATN grammar for the graphics/circuitboard domain.

The user can for instance request from the system:

```
show me all multipliers
show me D1M1
show me all faulty adders
please display D1
```

and many others, where D1 is the name of a device, and D1M1 the name of its first multiplier.

Diagnostic Reasoning

The diagnostic reasoning of VMES follows a simple control structure. It starts from the top level of the structural hierarchy of the device and tries to find output ports that violate an expectation. "Violated expectation" is defined as a mismatch between the expected (calculated) value and the observed (measured) value at some output. Though the target domain of VMES is digital circuit boards, we observed that in real life most of the electronic boards contain some simple analog components such as resistors and transformers. Therefore, for practical consideration, some components are allowed to have a tolerance when their outputs are being checked for violated expectations. The tolerance information is associated with the instantiation rule as depicted before.

After violated expectations are detected, the system uses the structural template to find a subset of components at the next lower

hierarchical level which might be responsible for the bad outputs. Then instantiation rules are activated to instantiate the suspects, and the suspects are ordered by some criteria for further investigation. Suspect ordering criteria will be discussed later. The diagnostic process is then continued on the instantiated suspicious parts. A part is declared faulty if it shows some violated expectation at its output port and it is at its intended maintenance level as described in the structural template or it is at the bottom level of the structural hierarchy and no further diagnosis is possible.

A small set of SNePS rules is activated at every stage of the diagnosis. For example, three rules are activated when reasoning about a possible violated expectation of a specific port of a device. One rule is to deduce the measured value of the port. A measured port value can either be deduced from wire connections or requested from the user. A similar rule is activated for the calculated value, and the last rule is used to compare the two values to decide if there is a violated expectation. The last rule is shown in Figure. 8 in both SNePS code and in *English*.

Suspects are first sorted into sublists by global criteria called fault possibilities. Fault possibility is determined by evaluating the suspects against the wholistic current situation, which is the current test results. For the current implementation, there is only one global *criterion: a suspect has higher fault possibility if it contributes to more* vio-expct output ports. Suspects within each sublist are then sorted by some local criteria called fault potentialities. Fault potentiality is a measure of the rate a particular type of component may fail. It is independent of the environment, only depending on the component type. (It may also depend on the lot number of the component, but so far we do not treat such details.) The ideal fault potentiality data for our domain is the thermal analysis data of the components. Due to the unavailability of the thermal analysis data, it is now implemented as an index ranging form 1 to 3. Component types with no stored fault potentiality data default to 2.

VMES does not make the single fault assumption. The system incorporates the user's judgement by offering him an opportunity to terminate the diagnosis session whenever a faulty part is located. The user can choose to continue the investigation of the remaining suspects if he feels that more faults are possible or if he would merely like to make sure other suspects have no problems.

Graphical Interface

As mentioned in an earlier section, there is a part of the VMES system that permits the user to graphically trace the whole reasoning process. This is done by a function called *display*, that retrieves knowledge about how objects look and how they are located from the network, and computes and creates a graphical representation from this knowledge. This method is analogous to the generation of natural language from a knowledge base, a widely accepted AI technique.

The reasoning part calls display with the name of the object that should be displayed, possibly with one or more of a number of options. We will give a quick review of the possible options; more details can be found in [Shapiro86a].

It is possible to select how may levels in the part hierarchy should be displayed. Objects can be shown blinking, and they can be blown up or shrunk to fill the screen or a predefined window optimally. It is possible to create two pictures, a detailed picture of an object and a picture of the "environment" of that object. The user has only to specify the object itself; the environment is retrieved from the part hierarchy by searching upwards.

A sub-option of the environment option permits the user to limit how may levels up in the part hierarchy is searched. The selection of what to show can be limited not only by the number of levels, but also by the number of parts or according to an approximate cognitive complexity which we are simulating by counting the number of graphics primitives visible on the screen.

Conclusions

The representation scheme described in this paper has been used to represent several devices, including several multiplier/adder boards and a six-channel PCM (Pulse Code Modulation) board for telephone communication. VMES has been successful in isolating the faults on these boards. A typical example is that VMES identifies an inverter on a PCM6 board as a faulty part, which actually accounts for the simultaneous malfunctioning of the two channels it affects, in the early stage of diagnosis. Though VMES has no capability to conclude that it is the only fault on the board, the suspect ordering criteria help the system to decide which suspect is to be checked first. The result shows that the representation scheme, along with an expandable component library leads to several important advantages: compact representation and system efficiencies in both system development and operating phases.

```
(build
    avb ($p $vc $vm $tr)
    &ant ((build port *p value *vc source calculated tolrnc *tr)
          (build port *p value *vm source measured))
    cq (build
          min 1 max 1
          arg (build name: THEY-MATCH p1 *vc p2 *vm
                    tolrnc *tr)
          arg (build port *p state vio-expct]
```

In English:

If the calculated and measured values of port p are known as vc & vm, one and only one of the following statements is true:
(1) vc and vm agree;
(2) port p displays a violated expectation.

Figure 8. A diagnostic rule.

We first claim that a clear distinction between the two abstraction levels of an object is desirable. The separation leads to system efficiency since the knowledge at the two abstraction levels are used at different stages of diagnosis. Level-1 information is used for detecting violated expectations, and level-2 information is used for suspect generation. To mix these two levels together will cause representation overhead and hamper system performance.

The use of the passive structural templates, which are never executed, to represent the substructure of objects of a component *type* has advantages over a procedural representation which uses a procedure or an instantiation rule for it [Davis83a, Shapiro86a]. Whenever it is necessary to reason about the substructure of an object, it is carried out on the unique structural *template* for the component *type* of the object. Only the sub-parts that require further examination will be instantiated (by the proper instantiation rules for them). Unlike the structural template representation, a procedural representation is used to instantiate "all" sub-parts of an object, and then the reasoning is carried out over the resulting substructures. This leads to serious system inefficiency due to representation explosion and resource waste caused by unnecessary object instantiation.

The most important feature of VMES is its versatility. VMES can easily be adapted to new devices by merely adding the structural and functional information of the "new" component *types* to the component library. A new component *type* is defined as a component *type* which has not previously been described to the component library. The new device itself is a new component type by our definition. The effort required to adapt the system to new devices should be minimal since digital circuit devices have a lot of common components, and the structural and functional description are readily available at the time a device is designed.

We have found at presentations that the graphics interface considerably improves the understandability of the reasoning process of the system. The use of a knowledge based graphics system promises to simplify the creation of graphics for new devices, in this way aiding the versatility of the system. The common representation for diagnosis, graphics and a number of natural language tools has aided us in adding a natural language component to the system, and in this way strengthened our belief in the usefulness of a knowledge based graphics system as a natural interface for a user friendly maintenance expert system.

As a *spin off*, we have found limitations in the classical part hierarchy and inheritance mechanisms, and we have started to work on a module for *Intelligent Machine Drafting* as a part of our system.

References

Brown81a.
D. C. Brown and B. Chandrasekaran, "Design Consideration for Picture Production in a Natural Language Graphics System," *Computer Graphics* 15(2) pp. 174-207 (July 1981).

Davis83a.
R. Davis and H. Shrobe, "Representing Structure and Behavior of Digital Hardware," *Computer*, pp. 75-82 (Oct. 1983).

Davis84a.
R. Davis, "Diagnostic Reasoning Based on Structure and Behavior," *Artificial Intelligence* 24 pp. 347-410 (1984).

Eades86a.
P. Eades and A. J. Lee, "Perception of Symmetry," 67, University of Queensland, St. Lucia (March 1986).

Friedell84a.
M. Friedell, "Automatic Synthesis of Graphical Object Descriptions," *Computer Graphics* 18/ N.3(1984).

Genesereth.
M. R. Genesereth, "The Use of Design Descriptions in Automated Diagnosis," *Artificial Intelligence* 24 pp. 411-436 (1984).

Hartley84a.
R. T. Hartley, "CRIB: Computer Fault-finding Through Knowledge Engineering," *Computer*, pp. 76-83 (March 1984).

Hunnman84a.
M. Hunnman and P. Schefe, "The Design of SWYSS, a Dialgue System for Scene Analysis," in *Natural Language Communication with Pictorial Information Systems*, ed. Leonard Bole, (1984).

Shapiro79a.
S. C. Shapiro, "The SNePS Semantic Network Processing System," pp. 179-203 in *Associative Networks: The Representation and Use of Knowledge by Computers*, ed. Nicholas V. Findler,Academic Press, New York (1979).

Shapiro82a.
S. C. Shapiro, "Generalized augmented transition network grammars for generation from semantic networks," *The American Journal of Computational Linguistics* 8(1) pp. 12-25 (1982).

Shapiro86a.
S. C. Shapiro, S. N. Srihari, M. R. Taie, and J. Geller, "VMES: A Network-Based Versatile Maintenance Expert System," pp. 925-936 in *Proc. of 1st International Conference on Applications of AI to Engineering Problems.*, Southampton, U.K. (April 1986).

Shortliffe76a.
E. H. Shortliffe, *Computer-Based Medical Consultations: MYCIN*, American Elsevier/North Holland, New York (1976).

Taie86a.
M. R. Taie, S. N. Srihari, J. Geller, and S. C. Shapiro, "Device Representation Using Instantiation Rules and Structural Templates," pp. 124-128 in *Proc. of Canadian AI Conference - 86*, Montreal, Canada (May 21-23, 1986).

Zydbel81a.
F. Zydbel, N. R. Greenfeld, M. D. Yonke, and J. Gibbons, "An Information Representation System," *IJCAI - 81*, (1981).

Biographies

Stuart C. Shapiro
Department of Computer Science
226 Bell Hall, Amherst Campus
State University of New York at Buffalo
Buffalo, NY 14260, USA

Stuart C. Shapiro received the S.B. degree in mathematics from MIT in 1966, and the M.S. and Ph.D. degrees in computer sciences from the University of Wisconsin, Madison in 1968 and 1971, respectively.

He currently holds the positions of Professor and Chairman of the Department of Computer Science at the University at Buffalo, where he has been since 1977. In 1971, he was a Lecturer in Computer Sciences at the University of Wisconsin, Madison. Between then and going to Buffalo, he was at the Computer Science Department of Indiana University, as Assistant and Associate Professor. In summer, 1974, he was a Visiting Research Assistant Professor at the University of Illinois at Urbana-Champaign.

Dr. Shapiro's research interests are in artificial intelligence, natural language processing, knowledge representation, and reasoning. He is editor of *The Encyclopedia of Artificial Intelligence* (John Wiley & Sons, forthcoming), the author of *Techniques of Artificial Intelligence* (D. Van Nostrand, 1979), *LISP: An Interactive Approach* (Computer Science Press, 1986), and over 60 technical articles and reports. He has served as a consultant on Artificial Intelligence for several companies, as department editor of the journal, *Cognition and Brain Theory* for Artificial Intelligence, and has served on the editorial board of the *American Journal of Computational Linguistics*.

Dr. Shapiro is a member of the ACM, the IEEE, the ACL, the Cognitive Science Society, and the AAAI. He is listed in *American Men and Women of Science*, and in *Who's Who in Artificial Intelligence*.

Sargur N. Srihari
Department of Computer Science
226 Bell Hall, Amherst Campus
State University of New York at Buffalo
Buffalo, NY 14260, USA

Sargur N. Srihari received his Bachelor's degree in Electrical Communication Engineering from the Indian Institute of Science in 1970 and Ph.D. degree in Computer and Information Science from the Ohio State University in 1976. He is presently a tenured associate professor in the Computer Science department at SUNY Buffalo. In summer 1979, he was a visiting Research Assistant Professor at the University of Waterloo in Canada. His current research interests are in artificial intelligence, and in particular, knowledge-based systems for diagnosis and computer vision.

Dr. Srihari has published over 60 journal articles and conference papers. He is the author of *Computer Text Recognition and Error Correction* (IEEE Computer Society Press, 1984). He is presently an associate editor of the Pattern Recognition journal. He is on the advisory board of the Second AI Applications to Engineering Problems Conference to be held at MIT in 1987.

Dr. Srihari is a senior member of IEEE, and member of the Association for Computing Machinery, American Association for Artificial Intelligence and Pattern Recognition Society.

James Geller
Department of Computer Science
226 Bell Hall, Amherst Campus
State University of New York at Buffalo
Buffalo, NY 14260, USA

James Geller is a doctoral candidate in the department of Computer Science at the State University of New York at Buffalo. He received a "Dipl. Ing." in Electrical Engineering from the Technical University Vienna, Austria in 1979 and the MS in Computer Science from the State University of New York at Buffalo in 1984. His current research interests are in artificial intelligence, and in particular, knowledge representation for pictorial domains.

Mingruey R. Taie
Department of Computer Science
226 Bell Hall, Amherst Campus
State University of New York at Buffalo
Buffalo, NY 14260, USA

Mingruey R. Taie is a doctoral candidate in the department of Computer Science at the State University of New York at Buffalo. He received a BS in Mechanical Engineering from the National Taiwan University in 1978 and the MS in Computer Science from the State University of New York at Buffalo in 1984. His current research interests are in artificial intelligence, and in particular, knowledge representation and fault diagnosis.

Appendix A2:

"Modeling Connections for Circuit Diagnosis"

Taie, M.R.
Srihari, S.N.

MODELING CONNECTIONS FOR CIRCUIT DIAGNOSIS†

Mingruey R. Taie and Sargur N. Srihari

Department of Computer Science
State University of New York at Buffalo
226 Bell Hall
Buffalo, NY 14260
Net Address: taiemr%buffalo-cs@CSNET-RELAY.ARPA

ABSTRACT

Model-based circuit diagnosis isolates the faulty components of a malfunctioning electronic device by reasoning on the basis of structural and functional description of the device. In this paper, we argue that explicit representation of wires and points of contact (PCONs) is necessary for diagnosing faults of circuit connections. The traditional model of a wire as a uni-directional module is inappropriate, because it ignores its bi-directional nature, and it does not include PCONs. A new model of wires and PCONs and the corresponding semantic network representation are presented. A wire is modeled as a bi-directional module to preserve its physical property, and its uni-directional design intention is retained by the connection mechanism. A deliberate component connection mechanism by either forming a PCON from two different ports or superimposing two same ports together is devised and implemented. Examples of using this model for circuit diagnosis are shown, and its limitations are discussed.

1. INTRODUCTION

A design-model-based fault diagnosis system reasons on the basis of structural and functional descriptions of a device. The performance of such a system, in terms of effective diagnosis and system generality, is dependent largely on the device model. The structure of a device is usually modeled hierarchically, and components are modeled as modules with I/O ports [2, 4, 8, 10]. This allows the system to focus on relevant parts of the device. Wires and wire connections, which, according to domain experts, are among the major causes of device malfunction, have not received the deserved attention in previous work.

A wire is often modeled as a common module with I/O ports, just like other components [2, 4, 10]. By doing this, a wire is implicitly assumed to be uni-directional, since I/O ports are uni-directional by convention. This reflects the design intention of the wire in the device, but conflicts with the physical reality of a wire — although both are important in fault diagnosis. Though this kind of model is in general good for components such as inverters and IC chips, its use for modeling wires presents two problems. First, the "implicitness" of the assumption itself is dangerous for fault diagnosis systems, because it makes the systems incapable of adjusting themselves to different diagnostic situations [1]. Second, treating wires as uni-directional modules makes it impossible to diagnose some wire-related faults, as shown in the following example.

A wire which emerges from point A for some length and then forks to points B and C, which is shown in Figure 1, is represented as

> (run-wire (A) (B)) (run-wire (A) (C)) in [2],
> where run-wire is a function that is used to create a wire from its first argument to the second; or
>
> (CONN (A) (B)) (CONN (A) (C)) in [4],
> where CONN is a two-argument predicate in some kind of logic programming language, and it asserts that there is a connection from its first argument to the second; or
>
> (build from A to (B C)) in [10],
> where build is a command which builds a node with a "from" arc pointing to A and two "to" arcs pointing to B and C respectively, which node asserts that A is connected to B and C.

Here A, B and C represent the three wire ends (wire ports) of the wire; the actual representation of a wire end in some systems may be more complicated than what we show here. All these representations treat a single piece of wire as two separate wires, and show no direct connection between point B and point C. Thus a short (bridge) to ground at point C cannot be noticed at point B, which is guaranteed to have a value supplied at point A in these models. This may mislead a fault diagnosis system to conclude that the component which is connected to the wire at point B is faulty, since it shows some bad output and its input (at point B) is assumed to be intact.

Another problem concerning wire connections, which has also been long ignored by researchers in fault diagnosis, is the problem associated with the contact points in an electronic device. According to experts in the circuit domain, a "bad contact",

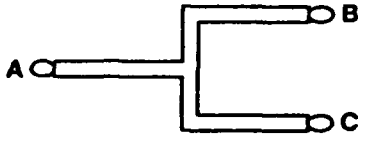


Figure 1. A three-end wire.

† This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB DC 20332 under Contract No. F30602-85-C-0008, which supports the Northeast Artificial Intelligence Consortium (NAIC).

i.e., a problem in wire connectivity, is a common fault of electronic devices. A bad contact in connection is usually misperceived as a bad wire or a bad chip in fault diagnosis systems. A loose pin and socket means an improper connection between a pin and its host socket, but not a bad chip, which the pin belongs to, or a bad wire, which the socket belongs to. In this paper we argue that "point of contact (POCON)", which is missing in the existing literature, should be modeled and explicitly represented for effective and complete fault diagnosis.

In the following sections, we first review the common model of components as a module with I/O ports, and then we introduce a new model for wires and POCONs along with some implementation details. Examples of using this new model for circuit diagnosis are shown, and its limitations are discussed.

2. A MODEL FOR COMMON COMPONENTS

In general, components in a device can be categorized as wires and non-wires. We refer to non-wires, i.e., the components other than wires, as "common components", and treat wires as special components.

Hierarchical models of devices are used to abstract devices at different levels of detail. A common component at any abstract level is in turn abstracted at two levels. At the level-1 abstraction, a common component is represented as a module (a black box) with I/O ports and associated functions. At the level-2 abstraction, subparts and wire connections of the component are envisioned. Figure 2 is the pictorial illustration of the model for common components.

In our system, instead of explicitly representing every part of a device at all hierarchical levels, we use an expandable component library to maintain descriptions of component "types", and parts are instantiated dynamically as needed. The significant part of this model is the clear distinction between level-1 and level-2 abstraction, which are represented in our system implemented in SNePS, the Semantic Network Processing System [7], as an instantiation rule and a structural template respectively. This model along with the expandable component library has advantages for system efficiency and versatility. More details can be found in [10].

We investigate device modeling from a knowledge representation view and conclude that there are two kinds of object in device representation by its nature. The first kind is the object that can be isolated from its environment when it is being investigated, examples are "B1" in "the PCM6 board, B1, is suspicious", "B1-IX3" in "the transformer, B1-IX3, is faulty", and "B1-W12" in "the wire, B1-W12, is interrupted". We call this kind of object a "stand-alone object", which includes both common and special components. The second kind of objects are those which are always associated with some other object(s). Examples are I/O ports of components and POCONs in connections. Please note that I/O ports are imaginary objects which conveniently represent the "places" where data (signals) flow into or out of a component, and POCONs represent the "relative physical relation" between two other objects. This kind of object is termed "parasitic object" since it is not a real object, and its existence always depends on the object(s) it is associated with. Figure 3 shows the semantic network representation for these two kinds of objects as well as the hierarchical representation of a device. Wires, component connections, and POCONs are discussed in following sections, and thus are not included in Figure 3.

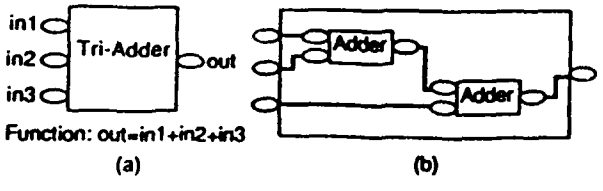


Figure 2. A model for common components. A tri-adder is used as an example. (a) level-1 abstraction. (b) level-2 abstraction.

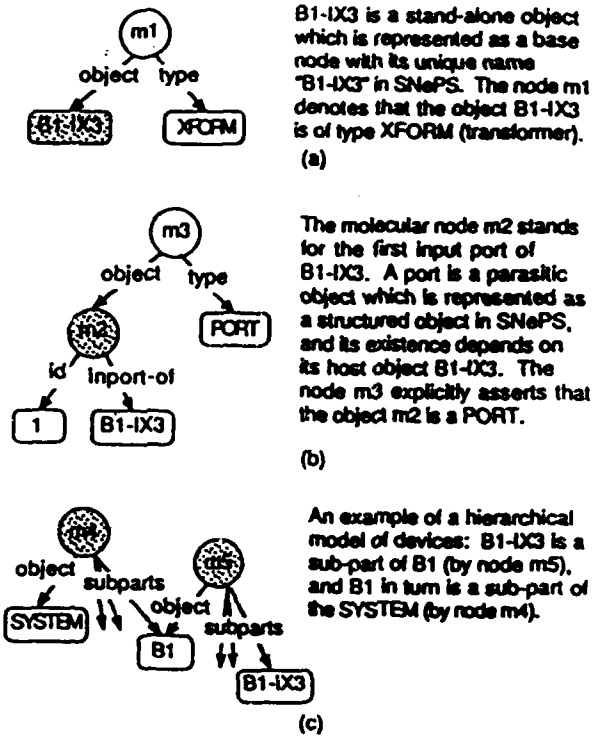


Figure 3. Objects and hierarchical model of devices: (a) stand-alone object; (b) parasitic object; (c) hierarchical device model.

3. A MODEL FOR WIRES

In most previous work, a wire is modeled in a manner similar to other common components as modules with I/O ports, [2] or not explicitly modeled at all [4,10]. We have shown in the introduction that this is inadequate, and here a new model for wires is presented. Wires are special components in several aspects: a wire is merely a piece of metal in some form: twisted copper threads wrapped by plastics or strips on a printed circuit board; the sole function of wires is to transmit signal (voltage and/or current) from one point to another; and it is maintained at all maintenance levels — in contrast, a common component such as a board is usually replaced only in the field, and a chip is replaced at a shop (depot).

In the work presented here, a wire is still modeled as a module, but the traditional uni-directional I/O ports are discarded. Instead, the wire ends are represented as bi-ports, a new term that stands for bi-directional ports, which allow signals to flow in either direction. The semantic network representation of the wire model is shown in Figure 4 with the three-end wire in Figure 1 as an example. The example wire is explicitly represented as a whole piece — a module with three bi-ports. Now the behavior of a wire can easily be represented by a rule asserting that all bi-ports of a wire have the same value (with ground dominant). The rule concerning the behavior and diagnosis of wires will be shown later. This simple model of a wire reflects its physical properties and enables correct simulation and fault diagnosis of electronic circuits.

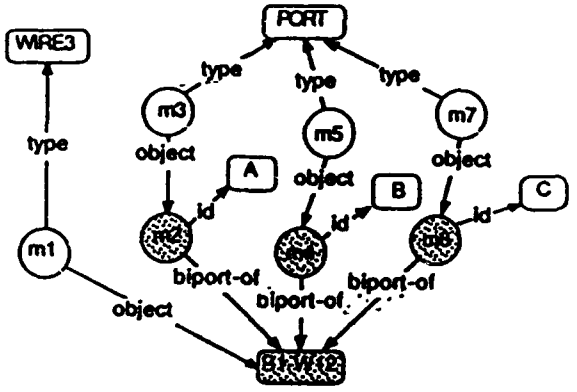


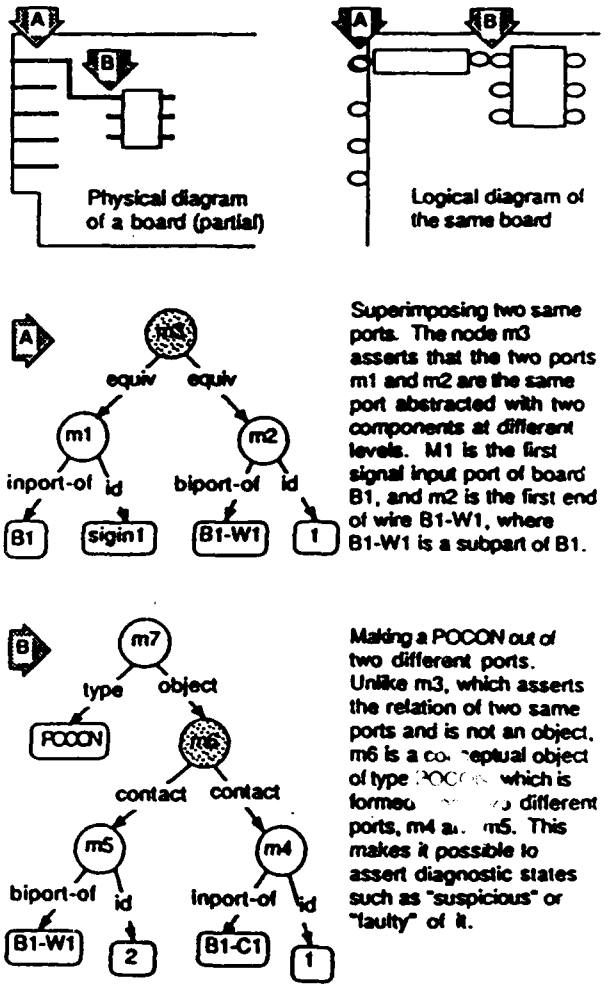
Figure 4. A model for wires. The three-end wire from Figure 1 is used as an example, and named B1-W12. WIRE3 is the component type of three-end wires. The nodes m2, m4 and m6 represent the three wire ends (bi-directional ports) of the wire B1-W12 with "id"s as A, B and C respectively. The nodes M3, m5 and m7 assert that m2, m4 and m6 are objects of PORT type.

As mentioned before, both bi-directional physical reality and uni-directional design intention of wires are important: the bi-directional property is useful in diagnosing some wire related faults; and the uni-directional property is good for fast generation of suspects by tracing back the part connection from bad outputs. It seems that our wire model only preserves the physical reality, but loses the design intention of a wire. The design intention ought to be presented in some way. A close investigation reveals that our wire model as module with bi-ports suffices. This model has the bi-directional property via the use of bi-ports. It also has the uni-directional property, though not explicitly represented in the model itself, via the connections with other components — a bi-port connected to an out-port serves as an in-port, and a bi-port connected to an in-port serves as an out-port. This improves system efficiency by saving the representation overhead of explicitly representing the uni-directional design intention in the wire model itself.

4. CONNECTION AND POCON

The explicit representation of points of contact (POCONs) has not been reported in the literature. For instance, Davis [2] deals with connections by superimposing the ports of two connected components, say, a wire and a chip. In this case, there is no way to find a faulty POCON, because there is no POCON at all. In order to catch this kind of problem, POCONs should be modeled and explicitly represented.

Unlike components such as chips and wires, POCONs are the "relative physical relation" between two ports of two components, which are "absolute physical entities", and thus POCONs cannot be modeled as physical components. Nevertheless, since the function of a POCON is to transmit signal between its two sides, it is reasonable to model a POCON as a logical component, and then we can assert properties such as "intact", "suspicious", and "faulty" of it. Like a port, a POCON is a parasitic object. A POCON is something that can not exist by itself, and is always referred to by the two ports with which it is associated. The semantic network representation of a POCON is shown in Figure 5.



Superimposing two same ports. The node m3 asserts that the two ports m1 and m2 are the same port abstracted with two components at different levels. M1 is the first signal input port of board B1, and m2 is the first end of wire B1-W1, where B1-W1 is a subpart of B1.

Making a POCON out of two different ports. Unlike m3, which asserts the relation of two same ports and is not an object, m6 is a conceptual object of type POCON which is formed by two different ports, m4 and m5. This makes it possible to assert diagnostic states such as "suspicious" or "faulty" of it.

Figure 5. POCONs and connection mechanisms.

There are actually two different connections needed in modeling electronic devices in a hierarchical way. One is to link two ports which are actually the same port abstracted at two different hierarchical levels. An example is that the port "in1" is a port of the tri-Adder in Figure 2(a), and it is also a port of the short wire in the left upper corner of Figure 2(b). Note that the tri-Adder and the wire are at different hierarchical levels, but they share the same port. This port is represented as two entities: one associated with the tri-Adder, the other with the wire. This provides the connections across hierarchical levels, and thus makes the transition of diagnosis or device simulation from one level to another possible. Since the two ports are actually the same one, this link is implemented by a "dual-equiv" semantic network caseframe,† which has the same effect as "superimposing" two ports [5]. (And this is the only right place for superimposition.) The other connection is to link components at the same hierarchical level together. Instead of superimposing two ports, we use a "dual-contact" semantic network caseframe to link the two ports by forming the representation of a POCON. The use of POCON preserves both the logical sense of connection and the physical sense of contact. Figure 5 illustrates these two kind of connection mechanisms.

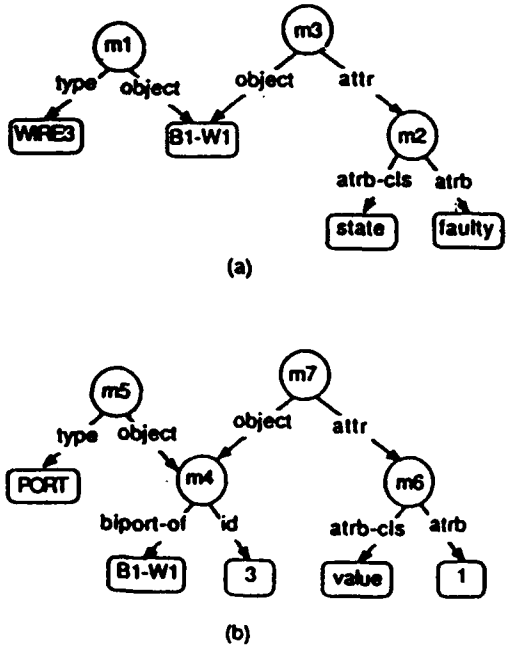


Figure 6. Semantic network representation for (a) diagnostic state associated with an object; and (b) value associated with a port.

† The "dual-equiv" semantic network caseframe is used in propositional semantic networks to link two intensionally distinct but extensionally equivalent objects together. See [5] for more detail.

5. FAULT DIAGNOSIS

In this section, we first show how a value and/or a state is associated with an object in Figure 6. Then, the methods of locating a faulty wire and a bad POCON are described. The details of the diagnosis by our experimental system VMES (Versatile Maintenance Expert System) are discussed in [8] and [10], and thus only the diagnosis concerning wires and POCONs is presented here.

All wires have the same function — transmitting values (signal as voltage) from one point to another. Though wires may have different numbers of wire ends, they all show the same behavior — the values at all ends of an intact wire are equal. It is usually necessary to simulate the behavior of a common component by calculating its outputs from its inputs using its functional description [3, 4, 8, 10]. Unlike for common components, there is no need to simulate the behavior of a wire. A simple rule which states that a wire is faulty if it has different values at its ends suffices. The rule we use to diagnose wires is shown in Figure 7.††

A POCON has similar function to a wire — it transmits a signal from one side to the other. Remember that our model for POCON is a logical object which is represented as a structured node with two "contact" arcs pointing to two ports of

In SNePSUL (SNePS User Language):

```
(build
 avb  ($p1 $p2 $w $v1 $v2)
 &ant ((build object *p1 bi-port-of *w)
       (build object *p2 bi-port-of *w)
       (build object *w type WIRE)
       (build object *p1
              attr (build atrb-cls value atrb *v1))
       (build object *p2
              attr (build atrb-cls value atrb *v2))
 cq   (build object *w
              attr (build atrb-cls state atrb faulty)
```

In English (not line by line translation):

IF p1 and p2 are different bi-ports of wire w, and
v1 and v2 are the values of p1 and p2, and
v1 is not equal to v2
THEN the wire w is faulty

Figure 7. Rule for diagnosing wires††

†† Under the UVBR (Unique Variable Binding Rule) of SNePS [9], different variables can not bind to a same value. This has the advantage of better reasoning efficiency by saving some antecedents of a rule, and by eliminating redundant variable bindings. In the rule we show here, there is no need to explicitly specify that p1 and p2 are two different ports as an antecedent. Note that the requirement of p1 and p2 being different ports is shown in the English translation of the rule. Moreover, there is no antecedent for checking if v1 and v2 are equal or not, because, under UVBR, the last two antecedents of the rule can be simultaneously satisfied only if v1 and v2 have different values.

two components. Therefore, a bad POCON can be defined as a POCON of which the two associating ports have different values. Note also that a POCON is only a conceptual object which represents a physical relationship between two ports of two components. It has no port of its own, and whenever the value of a port of a POCON is requested, the port is identified as a port associated with its host component rather than the POCON. In VMES, we currently treat the locating of a bad POCON as a by-product of checking components. This is done by the following method: whenever a port value is acquired, the system finds another port which forms a POCON with the first port, and if the second port has a value which is different from the value of the first port, then the system concludes that the POCON is bad. This is somewhat similar to the way a human expert diagnoses electronic circuits in that he measures some port values to check a component, but the result of the measurement stimulates him to conclude that the fault is on a nearby POCON rather than on the component he intends to check [6].

By using the model of wires and POCONs and the diagnostic rule and method described here, VMES has successfully located a broken (interrupted) wire and a bad contact point on a malfunctioning six channel pulse coded modulation board for telephone communication. Part of the screen output for a diagnosis session is shown in Figure 8.

```
(diagnose B1 PCM6)
@@@@@ diagnose B1: searching vio-expct...
@> Test Configuration for PCM6 Board:
@> ----------
@> 1. connect pcmout to pcmin for every channel.
@> 2. synchonize T-STROBE/SHIFT and R-STROBE/SHIFT
@@@@@ vio-outps found: (m36 m34)
(m36 (id (sigout1)) (outport-of (B1)))
(m34 (id (sigout3)) (outport-of (B1)))
@@@@@ suspects created:
   (B1-NOTG.r0 B1-NOTG.t0 B1-WR1 B1-WR2 B1-WRS
    B1-WT1 B1-WT2 B1-WTS B1-PC1 B1-PC3 B1-IX1
    B1-IX3 B1-OX1 B1-OX3 B1-W11 B1-W12 B1-W13
    B1-W14 B1-W31 B1-W32 B1-W33 B1-W34)
@@@@@ diagnose B1-NOTG.r0: searching vio-expct...
what's the value of port
(m731 (id (out)) (outport-of (B1-NOTG.r0)))
* [value]/nil?   1
what's the value of port
(m696 (id (in)) (inport-of (B1-NOTG.r0)))
* [value]/nil?   0
@@@@@ B1-NOTG.r0 shows no problem
. . . . . .
@@@@@ diagnose wire: B1-WR1
what's the value of port
(m761 (id (2)) (biport-of (B1-WR1)))
* [value]/nil?   0
what's the value of port
(m732 (id (3)) (biport-of (B1-WR1)))
* [value]/nil?   1
@@@@@ wire B1-WR1 is faulty
      by showing different values at wire ends
Terminate the diagnosis?
* y/n?  y
VMES manually terminated
@@@@@ I GOT THE FAULTY PARTS AS:
(B1-WR1)
Repair Order: fix B1-WR1 (type:WIRE3)
done
```

Figure 8(a). Locating a bad wire on a PCM6 board.

```
* (diagnose B1 PCM6)
@@@@@ diagnose B1: searching vio-expct...
  . . . . . .
@@@@@ vio-outps found: (m36 m34)
(m36 (id (sigout1)) (outport-of (B1)))
(m34 (id (sigout3)) (outport-of (B1)))
@@@@@ suspects created:
   (B1-NOTG.r0 B1-NOTG.t0 B1-WR1 B1-WR2 B1-WRS
    . . . . . .
@@@@@ diagnose B1-NOTG.r0: searching vio-expct...
  . . . . . .
@@@@@ diagnose wire: B1-WR1
what's the value of port
(m761 (id (2)) (biport-of (B1-WR1)))
* [value]/nil?   0
what's the value of port
(m746 (id (1)) (biport-of (B1-WR1)))
* [value]/nil?   0
what's the value of port
(m732 (id (3)) (biport-of (B1-WR1)))
* [value]/nil?   0
@@@@@ The pocon m733 between the ports
(m732 (id (3)) (biport-of (B1-WR1)))
(m731 (id (out)) (outport-of (B1-NOTG.r0)))
is faulty!
Terminate the diagnosis?
* y/n?  y
VMES manually terminated
@@@@@ I GOT THE FAULTY PARTS AS:
(m733
  (contact (m731 (id (out)) (outport-of (B1-NOTG.r0)))
           (m732 (id (3)) (biport-of (B1-WR1)))))
Repair Order: fix the contact point
done
```

Figure 8(b). Locating a bad POCON on a PCM6 board.

Though the rule shown in Figure 7 is good for locating interrupted wires, it cannot handle the bridge (short circuit) problem of wires. When a wire is bridged to ground or some other component, all wire ends still have the same value (with ground dominating). This makes the bridge problem difficult to diagnose. Nevertheless, this problem is not unsolvable. A bridged wire can be identified by observing a difference between the measured value at wire ends and the supplied value of the wire. The supplied value of the wire is the value which the wire is intended to transmit. The supplied value could be the calculated value of an output port of a component which is connected to the wire, or it could be from a requested test in which a value is actually supplied to one wire end from outside the board. Since wires are explicitly modeled and represented in our system, the necessary procedures and user interactions for carrying out the test can easily be implemented, and a bridged wire can be located A harder problem is how to decide if the bridge is on the wire or on some component which is connected to the wire. One way is to disconnect the wire and the component for separate test. Though this ensures that a bridge can be correctly located, this operation may be too costly and impractical. A more reasonable treatment is to ask the user to check it. Since human beings have much better visual capabilities than computers, they can detect most bridge faults by inspection if they are directed to the right area.

6. CONCLUSIONS

Explicit representation of wires and POCONs is necessary for diagnosing faults regarding connections. Our system, using a new model of wires and POCONs as well as a deliberate component connection mechanism of either forming a POCON from two different ports or superimposing two same ports together, is effective in circuit simulation and fault diagnosis. Our system has been successful in locating interrupted wires and bad contact points, and it has been shown that it could handle bridge problems of wires to some extent.

ACKNOWLEDGEMENT

The authors would like to thank Stuart C. Shapiro, James Geller, and Scott S. Campbell for their constructive comments on this paper and as coworkers on the VMES project. Stuart Shapiro's generous help in using SNePS is also appreciated. The authors would also like to thank Dale Richards and Norman Sturdevant for their help on the overall project.

References

1. R. Davis, H. Shrobe, W. Hamscher, K. Wiechert, M. Shirley, and S. Polit, "Diagnosis Based on Description of Structure and Function," pp. 137-142 in *Proc. of AAAI-82*, Morgan Kaufmann, Los Altos, CA (August 1982).
2. R. Davis and H. Shrobe, "Representing Structure and Behavior of Digital Hardware," *Computer*, pp. 75-82 (Oct. 1983).
3. R. Davis, "Diagnostic Reasoning Based on Structure and Behavior," *Artificial Intelligence* 24 pp. 347-410 (1984).
4. M. R. Genesereth, "The Use of Design Descriptions in Automated Diagnosis," *Artificial Intelligence* 24 pp. 411-436 (1984).
5. A. S. Maida and S. C. Shapiro, "Intensional Concepts in Propositional Semantic Networks," *Cognitive Science* 6(4) pp. 291-330 (1982).
6. J. Rasmussen and A. Jensen, "Mental Procedures in Real-Life Tasks: A Case Study of Electronic Trouble Shooting," *Ergonomics* 17 pp. 293-307 (1974).
7. S. C. Shapiro, "The SNePS Semantic Network Processing System," pp. 179-203 in *Associative Networks: The Representation and Use of Knowledge by Computers*, ed. Nicholas V. Findler, Academic Press, New York (1979).
8. S. C. Shapiro, S. N. Srihari, M. R. Taie, and J. Geller, "VMES: A Network-Based Versatile Maintenance Expert System," pp. 925-936 in *Proc. of 1st International Conference on Applications of AI to Engineering Problems*, Springer-Verlag, New York (April 1986).
9. S. C. Shapiro, "Symmetric Relations, Intentional Individuals, and Variable Binding," *Proceedings of IEEE*, (October, 1986).
10. M. R. Taie, S. N. Srihari, J. Geller, and S. C. Shapiro, "Device Representation Using Instantiation Rules and Structural Templates," pp. 124-128 in *Proc. of Canadian AI Conference - 86*, Presses de l'Université du Québec, Montréal, Canada (May 1986).

Appendix A3:

"Applications of Expert Systems in Engineering"

Srihari, S.N.

Applications of Expert Systems in Engineering

Sargur N. Srihari, Ph.D.
Department of Computer Science
State University of New York
at Buffalo

Abstract

Knowledge-based expert systems are computer programs for acquiring and manipulating knowledge for solving a range of problems in a given domain. Engineering is the use of specialized knowledge to solve real world problems. Thus there is a close match between the objectives of engineering and the capabilities of expert systems technology. This paper is an overview of the methodology of knowledge-based expert systems, their implementation, and their potential application in engineering. Methods of knowledge representation in first generation expert systems and in future generation expert systems are looked at. Some ongoing research on expert systems methodology at the State University of New York at Buffalo are described.

This work was supported by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-57000, and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under Contract No F 30602-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium.

1. Introduction

A human expert possesses knowledge about a domain, has an understanding of domain problems and has some skill at solving problems. An expert system for a given domain is a computer program that is capable of solving problems at a performance level comparable to that of most human experts in that domain.

Early successes of expert systems were in domains within the fields of medicine (MYCIN diagnoses bacterial infectious diseases) and the sciences (DENDRAL identifies chemical structures). More recently many applications have been found in traditional engineering domains. This is only natural in that engineering, defined broadly, is the use of specialized knowledge to solve real world problems. Examples of applications of expert systems in engineering are: configuring orders for computers (XCON configures VAX computer systems at Digital Equipment, similarly BEACON is used at Burroughs, CONAD at NCR, and DRAGON at ICL), trouble shooting problems (with diesel locomotives, telephone switching equipment, industrial fan problems, centrifugal pumps, disk drives, utility transformers), process control (chemical plant control optimization), etc.

The objective of this paper is to outline the methodology of first and future generation expert systems in the context of their applicability to engineering and to describe some of the ongoing research on expert systems methodology at the State University of New York at Buffalo.

1.1 Problem Characteristics

The characteristics that distinguish an expert system from other computer programs are that they perform intellectually demanding tasks at expert level performance, that they emphasize domain specific methods of problem solving over general algorithms of computer science, and that they provide explanations for conclusions reached or actions taken.

The kinds of problems that expert systems solve may be divided into those of *analysis* and those of *synthesis*. Analysis problems involve starting with large amounts of data and arriving at conclusions in summary form. Examples of analysis problems are interpretation and diagnosis. Interpretation involves classifying or describing data; thus a pattern recognition task such as determining crop categories in a satellite photograph is an expert interpretation problem. Diagnosis is the problem of determining from a set of observables (symptoms, physical findings, results of tests) the causes for the unusual manifestations. Diagnosis in the engineering domain is most often a problem of localization of faults using structural and functional models of the system. Synthesis problems involve beginning with primitive components and arranging or rearranging them to satisfy requirements. Examples of problems of synthesis are configuring a computer, designing a building, designing a mechanical or electrical component, etc.

1.2 Role of Knowledge

Expert problem solving seems to involve search through a judgemental knowledge base specialized to the domain. The knowledge can be either public (published definitions, facts, theories) or private (not in books). Private knowledge is typically in the form of rules of thumb, called *heuristics*, that are learned and refined over years of problem solving experience in that domain. In fact, the central task of building expert systems is that of elucidating and reproducing private knowledge. *Knowledge engineering* is the task of extracting human expert knowledge and organizing an effective representation. The organized knowledge, including heuristics, general models and causal models of behavior is a knowledge base.

2. Using Production Rules

The method of knowledge representation should have the following characteristics: capture generalization, be understood by people providing it, be easily modifiable, and be useful in a great many situations. Knowledge is represented in *first generation* expert systems in the form of a set of production rules. The basic form of a production rule is:

Rule R_n:
If $a_1, a_2, \ldots, a_m$
Then $b_1, b_2, \ldots, b_k$

where a_i are predicates (statements that can have true or false values) that are referred to as *antecedents* (also premises, patterns, conditions) and b_i are referred to as *consequents*. The consequents can either be *deductions* or *actions*. A deduction is inferred from facts about a given situation. An example of a deduction rule is (If (LIGHT ON) Then (CAN SEE)). Deductions are most common in diagnostic reasoning. An action rule changes one situation to another, e.g. (If (PUMP PRESSURE) > 300 Then (SHUT DOWN PUMP)). Action rules describe expert behavior in terms of available operations.

A rule can be viewed as a conditional

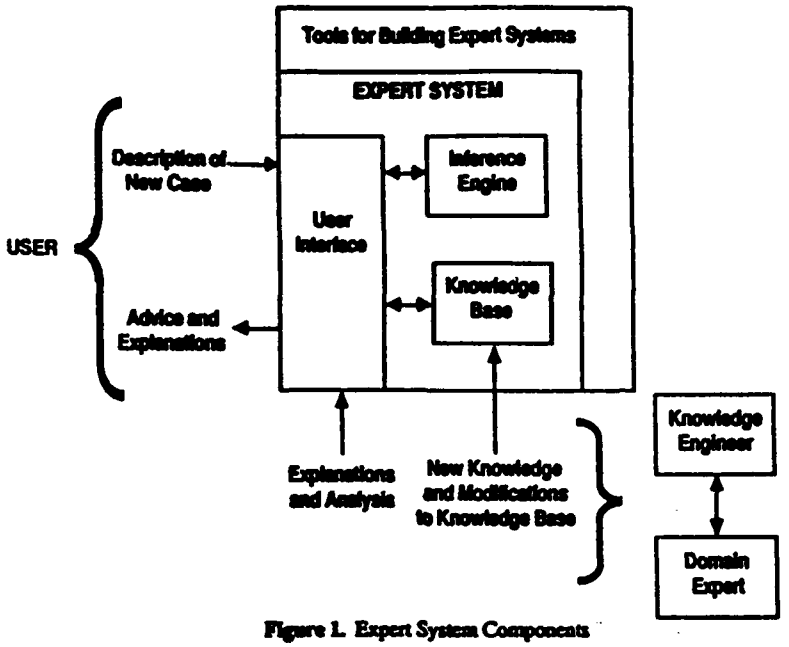


Figure 1. Expert System Components

statement, and the invocation of rules as a sequence of actions chained by *modus ponens*. According to modus ponens, if *A implies B* is true and *A* is true, then *B* is true. For example, given the rule "If x is human then x is mortal" and the fact "Socrates is human," by instantiating x to Socrates and applying modus ponens we have Socrates is mortal. More generally, if *A* implies *B* and *B* implies *C*, then *A* implies *C* — which is referred to as *syllogism*.

When all of the a_i are true, rule R_n is said to be triggered. A rule is selected from the set of triggered rules, or *conflict* set, using a conflict resolution strategy. When the consequents of the selected rule are performed, the rule is said to be *fired*. There are several strategies for selecting the rule for firing from the conflict set. Some of these are:

1) Specificity ordering — arrange rules whose conditions are a super-set of another rule.

2) Rule ordering — arrange rules in priority list; rule appearing earliest has highest priority.

3) Data ordering — arrange data in priority list; rule having highest priority data (condition) has highest priority.

4) Size ordering — rule having longest list of constraining conditions has highest priority.

5) Context limiting — activate (or deactivate) groups of rules at any time; thus there is less likelihood of conflict.

The choice of conflict resolution strategy is ad-hoc. Specificity ordering and context limiting strategies are more often encountered than others.

The different components of a production rule based expert system are shown in Figure 1. The *knowledge base* consists of the set of production rules, *context* is a workspace for the problem constructed by an inference mechanism from the data provided by the user and the knowledge base, and the *inference engine* modifies the context. Apart from the main modules an expert system should also be provided with a graceful *user interface*, an *explanation facility*, and a *knowledge acquisition* model.

2.1 Forward and Backward Chaining

In the case of synthesis systems the antecedents of the rules are conditions and the consequents are actions. The interpretation or control mechanism used by synthesis systems is as follows:

1) Collect rules whose *if* parts are triggered and select a rule using a *conflict resolution* strategy.

2) Do what the rules' *then* part says (fire).

In the case of analysis systems the antecedents of rules are either observed or derived facts and the consequents are new facts that are derived. This mechanism is said to *forward chain* the rules. The control mechanism used by analysis systems can be either forward or *backward* chaining. In backward chaining, a particular hypothesis is selected using some discipline. The rules are examined to see if the hypothesis is a consequent. If so, the antecedents of such form the next set of hypotheses. The procedure is continued until some hypothesis is false or all hypotheses are true based on the data.

Forward and backward chaining analysis systems are analogous to bottom-up and top-down control in general computer algorithms, e.g., compilers. Bottom-up analysis systems progressively refine the data to draw conclusions. Top-down analysis systems begin with an expectation of what the data could define and see if the data fits the expectation.

2.2 Certainty Computation

It is often useful or necessary to associate levels of confidence with rules as well as with antecedents and consequents. Thus the inference mechanism needs a method for, say, computing the confidence of the consequent given the confidence values of its antecedents and the confidence of the rule. One approach, based on probabilities, is as follows:

- The certainty of a rule's overall input is the product of the certainties associated with the rule's antecedents.
- The certainty of a rule's output is given by a single valued function having input certainty on one axis and output certainty on another.
- The certainty of a fact supported by several rules is determined by transforming certainties into related measurements, called uncertainty ratios, then transforming the certainty ratios back into a certainty.

2.3 Production Rule Languages

In principle, an expert system can be programmed in any programming language such as FORTRAN, C, or LISP. However, as with any complex system, the choice of the tool can influence the feasibility of constructing and/or modifying it. Production rule systems can be efficiently implemented using many different commercially available tools. Prominent among these are: EMYCIN, KAS, ROSIE, OPS5 and PROLOG, among other. Each of these languages/systems provide a built-in inference mechanism.

EMYCIN, a system derived from MYCIN[1], is particularly suited to diagnosis problems, provides backward chaining, and allows certainties between $^{\prime}1$ and $+1$ to be associated with data and conclusions. KAS has derived from PROSPECTOR, an expert system for geological exploration. It uses likelihood ratios for rule strengths and control can be both forward and backward. OPS5 offers generality in that it is easy to tailor the system to the domain but unlike EMYCIN and KAS it does not offer sophisticated front ends. OPS5 uses forward chaining exclusively. ROSIE uses English-like syntax but has no sophisticated data base structure. PROLOG is a logic programming language

that provides an inference mechanism that is limited to backward chaining. Many production rule systems, e.g., OPS5, are written in LISP (the list processing language most often used in artificial intelligence programming).

2.4 Rule Based Systems

Advantages of a rule-based system are: they enforce a homogeneous representation of knowledge, allow incremental knowledge growth through addition of rules and allow unplanned but useful interactions — knowledge can be applied when needed and not when the programmer predicts them (by the same token, a disadvantage is that the user can lose control).

Several problems of analysis and synthesis have been successfully handled by *first generation* expert systems which use judgemental knowledge of human experts in the form of a monolithic set of production rules. Among those that have been applied in engineering is XCON[2], a synthesis expert system for configuring Digital Equipment Corporation's VAX computers and CRIB[3], an analysis expert system for computer hardware fault diagnosis.

Experience with rule based systems has shown the following drawbacks: knowledge acquisition from domain experts — which is the process of *knowledge engineering* — is time consuming or difficult; all possibilities have to be explicitly enumerated; and they have almost no capability of system generalization.

3. Semantic Networks and Frames

The keystone to the success of expert systems is the effective representation of domain knowledge. Domain knowledge typically has many forms, including descriptive definitions of domain specific terms (e.g., "power plant," "pump," "flow," "pressure"), descriptions of individual domain objects and their relationships to each other (e.g., "P1 is a pump whose pressure is 230 psi"), and criteria for making decisions (e.g., "if the feedwater pump pressure exceeds 400 psi, then close the pump's input valve").

3.1 Semantic Networks

A semantic network is a method of knowledge representation (see Figure 2). Concepts are represented as nodes (circles) in the network and relations are represented as directed arcs (labeled arrows). A node-and-link net is not necessarily a semantic net, however. To be a semantic network there must be a way of associating meaning with the network. One way of doing this, called *procedural semantics*, is to associate a set of programs that operate on descriptions in the representation.

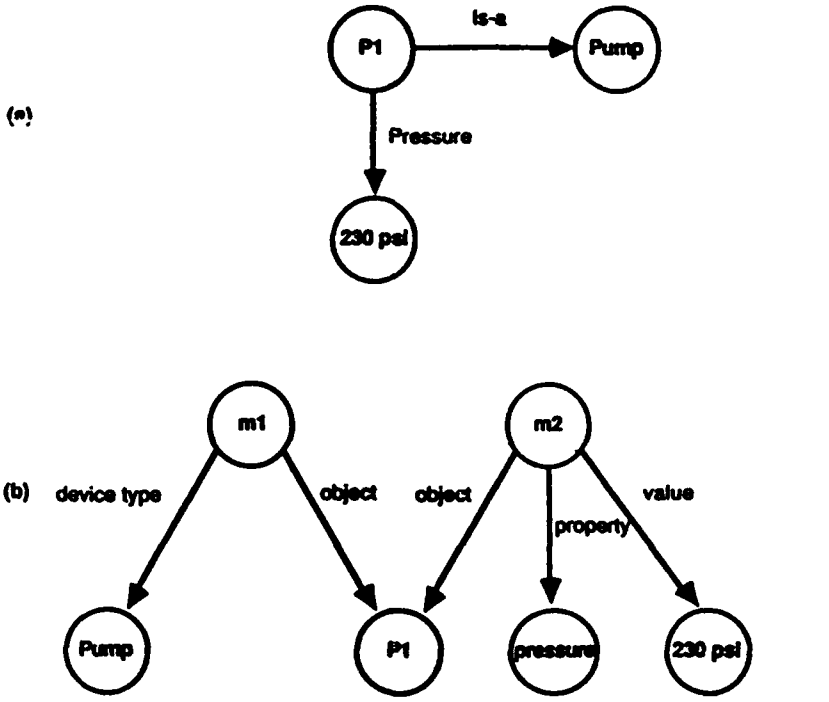
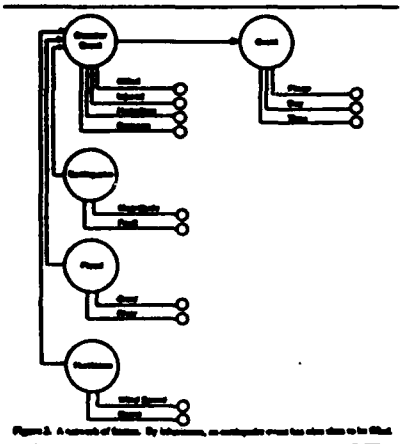


Figure 2. A semantic network consists of labeled nodes and arcs with associated methods of interpreting them: (a) network for "P1 is a pump at pressure 230 psi," (b) equivalent network in SNePS.

3.2 Frames

A frame provides a structured representation of an object or a class of objects. For example, one frame might represent an automobile and another a whole class of automobiles. In a sense a frame is a collection of semantic net nodes and slots that together describe a stereotyped object, act, or events (see Figure 3). Constructs are available in a frame language for organizing frames that represent classes into hierarchical taxonomies. In addition, special purpose deduction algorithms exploit the structural characteristics of frames to perform a set of inferences that extends the explicitly held set of beliefs to a larger, virtual set of beliefs.



3.3 Integrated Environments

Examples of systems that combine the advantages of both frame representations and production rule languages are LOOPS and KEE (Knowledge Engineering Environment). In these systems frames provide a rich structural language for describing the objects referred to in the rules. Frame taxonomies can also be used to partition a system's production rules.

SNePS is a semantic network processing system that allows relational knowledge as well as production rules to be represented in the form of a semantic network. SNePS is particularly appropriate when natural language interfaces are important.

Future generation expert systems will rely on a problem solving architecture such as the *blackboard*, which is a framework that allows several knowledge sources (or expert systems) to interact in the solution of a problem. An environment for such an approach is GBB (Generic Blackboard Development system).

4. Research at SUNY at Buffalo

Research is ongoing in the Department of Computer Science at SUNY at Buffalo on methodologies for expert systems that solve analysis problems. In particular we are exploring the design of knowledge-based systems for image interpretation (recognizing objects in complex environments) and for diagnosis. In the image interpretation area our work concerns how to coordinate a community of experts in achieving a common goal[4,5] and how to develop expert systems for utilizing spatial knowledge in image interpretation problems[6]. Our image interpretation methods emphasize the use of a blackboard architecture, frames, and production systems. In the diagnosis area our work centers on how to represent spatial and other model knowledge in diagnosis[7,8,9]. In the remainder of this section we focus only on our research in the area of diagnostic expert systems since it is a commonly encountered engineering problem.

In the diagnostic application of expert systems first generation systems can also be referred to as *symptom-based*, since the rules consist of mapping symptoms to intermediate or final conclusions. Expert systems based on direct mapping of observations to conclusion are said to have only *shallow* knowledge. Second and future generation diagnostic expert systems that utilize a model of the physicial system as an essential part of reasoning may be said to perform on the basis of *deep* knowledge. An important component of deep knowledge consists of design specifications of the device and hence the resulting systems may be said to be *specification-based*.

There are several ways of modeling a physical system. The analytical capability, flexibility and efficiency of the reasoning process can be expected to depend on the model selected. Two ongoing efforts on modeling and representation for diagnosis are described in the following.

4.1 Behavioral and Structural Models

A physical system can be modeled behaviorally and/or structurally. A behavioral model is a *logical* model and a structural model is a *physical* model. The method of modeling may be different at different levels of abstraction, depending on the task. The logical model is a functionally (causally) oriented abstraction where each logical component has a localized contribution to overall behavior. A physical model is a representation where each component is described in terms of its physical characteristics. In the process of modeling, functional (or logical) structure may be derived from an observation of the physical structure in a bottom-up manner. Two models of the same type have a hierarchical relation if one is an abstraction of the other. The logical and physical models of a physical system often correspond. For instance, the power train of an automobile consists of two functional (logical) components, the engine which provides power and the transmission which converts power to motion. These, in turn, correspond to the two physical components, the engine assembly and the transmission assembly. However, the two types of models sometime intersect. A digital circuit is functionally modeled by a logic diagram in which logical components such as "and gates" or "or gates" contribute locally to the whole circuit, but several functionally unrelated logical components may be contained in the same physical component, *viz.*, an integrated circuit chip. Our effort is to develop a theory of multi-level device representation within which structural, functional, as well as empirical (symptom) knowledge can be represented.

4.2 Versatile Maintenance

A theory and an implementation of a device representation scheme for versatile maintenance of digital circuits is being developed. Versatility of diagnosis is defined multidimensionally across a range of target devices, faults, maintenance levels, and user interfaces. Central to design model-based fault diagnosis is a device representation scheme with the following attributes: expressive power, ease of building the representation, compactness, generalization ability and expandibility, and ability to operate on the representation. Device structure is represented hierarchically to reflect the design model of most devices in the domain. Each object of the device hierarchy has the form of a module. Instead of representing all objects explicitly, an expandable component library is maintained, and objects are instantiated only when needed. The component library consists of descriptions of component *types* used to construct devices at all hierarchical levels. Each component is represented as an instantiation rule and a structural template. The instantiation rule is used to instantiate an object of the component type as a module with I/O ports and associated functional descriptions. Structural templates describe subparts and wire connections at the next lower hierarchical level of the component type. The implementation of VMES (Versatile Maintenance Expert System) is being done using SNePS.

5. Summary

There exists a close match between the capabilities of expert systems and the needs of engineering practice. Knowledge acquisition and representation is central to the design of expert systems. Production rules are the preferred method for representing knowledge in first generation expert systems. Second and future generation expert systems will integrate production rules with semantic networks, frames, and other knowledge representation schemes. Our research on expert systems methodology at SUNY at Buffalo is currently focused on problems of analysis such as image interpretation and fault diagnosis. In the diagnosis area the efforts are to develop a theory of multi-level device knowledge representation and an approach to versatile maintenance based on a semantic network representation.

References

1. E.B. Shortliffe, *MYCIN: Computer-Based Medical Consultations*, Elsevier, New York, 1976. A successful first generation expert system.
2. J. McDermott, "R1: A rule-based configurer of computer systems," *Artificial Intelligence*, vol. 19, no. 1, 1982. Precursor of XCON used at DEC.
3. R.T. Hartley, "CRIB: Computer fault-finding through knowledge engineering," *Computer*, vol. 17, no. 3, March 1984. An engineering application.
4. C.H. Wang and S.N. Srihari, "Object recognition in structured and random environments: locating address blocks on mail pieces," *Proceedings of the National Conference on Artificial Intelligence (AAAI-86)*, Philadelphia, August 1986, 1123-1127. A vision system using blackboard architecture under development for the U.S. Postal Service.
5. D. Niyogi and S. Srihari, "A rule-based system for document understanding," *Proceedings of the National Conference on Artificial Intelligence (AAAI-86)*, Philadelphia, August 1986, 789-793. A vision system being developed in collaboration with Xerox Webster Research Center.
6. R. Kumar and S.N. Srihari, "An expert system for interpreting cranial CT scans," *Proceedings of Expert Systems in Government Symposium*, McLean, October 1985, 548-557. A vision system done in collaboration with SUNY Radiology Department.
7. Z. Xiang, S.N. Srihari, S.C. Shapiro, and J.G. Chutkow, "Analogical and propositional representations of structure in neurological diagnosis," *Proceedings of IEEE Artificial Intelligence Applications Conference*, Denver, December 1984, 127-132. Explores spatial knowledge representation issues.
8. Z. Xiang and S.N. Srihari, "Diagnosis based on empirical and model knowledge," *Sixth International Workshop on Expert Systems*, Avignon, France, April 1986, 835-848. Describes a multi-level approach to diagnosis.
9. M.R. Taie, S.N. Srihari, J. Geller, and S.C. Shapiro, "Device representation using instantiation rules and structural templates," *Proceedings of the Canadian Artificial Intelligence Conference*, Montreal, May 1986, 124-128. A theory of device representation under development for VMES.

Further Reading

W.F. Clocksin and C.S. Mellish, *Programming in Prolog*, Springer-Verlag, New York 1981.

An introductory text on PROLOG which is more commonly used as an expert systems development tool in Europe.

R. Fikes and T. Kehler, "The role of frame-based representation in reasoning," *Communications of the ACM*, vol. 28, no. 9, September 1985. The originators of KEE describe it theoretical foundations.

C.L. Forgy, "OPS5 User's Manual," *Technical Report* CMU-CS-81-135, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1981. Readable introduction and manual.

P. Nii, "The blackboard model of problem solving," *AI Magazine*, vol. 7, Summer, 1986, 38-53. A tutorial on blackboards to coordinate several knowledge sources.

S.C. Shapiro, "The SNePS semantic network processing system," in N.V. Findler (ed), *Associative Networks: the Representation and Use of Knowledge by Computers*, Academic Press, 1979, 179-203. Introduction to SNePS by its inventor.

D. Sriram and R. Adey (eds.), *Applications of Artificial Intelligence in Engineering Problems*, Proceedings of the First International Conference, Southampton, England, Springer-Verlag, New York, 1986. Includes papers concerning applications of expert systems in a variety of engineering problems.

P.H. Winston, *Artificial Intelligence (second edition)*, Addison-Wesley, 1984. Excellent introductory text on artificial intelligence that contains a particularly lucid chapter on expert systems methodology.

Appendix A4:

"Device Representation and Graphics Interfaces of VMES"

Geller, J.
Taie, M.R.
Srihari, S.N.
Shapiro, S.C.

Device Representation and Graphics Interfaces of VMES

J. Geller, M.R. Taie, S.C. Shapiro, S.N. Srihari
Department of Computer Science, State University of New York at Buffalo, Buffalo, NY 14260, U.S.A.

ABSTRACT

The VMES project aims to create a device-model-based versatile maintenance expert system which assists the user in isolating specific faulty components or connections in a malfunctioning digital circuit. We describe a device representation formalism that supports the diagnostic reasoning of VMES and eases its adaptation to new devices. The salient feature of this scheme is the inclusion of both logical and physical structural descriptions of the target device. The two representations enable VMES to make efficient diagnostic judgements and to interact effectively with the user in performing repair and test. The user interface of VMES is treated as a separate area of scientific investigation. We describe the design and implementation of three interfaces, *viz.*, a graphics display interface, a graphics input interface, and a natural language input interface. The use of knowledge based interface technology has proven a rewarding area of research from the theoretical as well as the applied perspective.

INTRODUCTION

VMES is a device-model-based versatile maintenance expert system for the domain of digital circuits [7]. The objective of VMES is to interact with a maintenance technician (the user) to identify the specific faulty component or connection of a malfunctioning circuit. The versatility of VMES is multifold: across a wide range of devices, covering most possible faults, suitable for different maintenance levels, and providing an intelligent user interface. VMES uses a device-model-based approach since it is more general than the traditional empirical-rule-based approach [1, 2, 7].

VMES consists of five modules: the knowledge-base; the inference engine; the active database; the end-user interface; and the intermediate-user interface (Fig. 1). The knowledge-base is implemented as an expandable component library which contains component descriptions. The inference engine has the generic diagnosis knowledge of the domain, and uses the SNIP semantic network inference package of SNePS, the semantic network processing system, as its basis [5, 3]. An active database is created and updated throughout each diagnostic session to keep the instantiated objects and their associated diagnostic states and values. The end-user interface interfaces the maintenance technicians when carrying out a diagnostic session. The intermediate-user interface interfaces the engineers or senior technicians to update the knowledge-base for new devices. All these five modules are implemented on top of SNePS.

As knowledge engineering is to empirical-rule-based systems, device modeling/representation is the key to the success of a device-model-based fault diagnosis system, since knowledge about the structure and function of a device is the major knowledge source of reasoning in such a system. Consequently, our efforts are focused on the development of a device representation formalism for

versatile maintenance. All knowledge, whether of structural, functional, or graphical, is in a unified knowledge base (Fig. 2), which is easily expandable and is referred as a "component library".

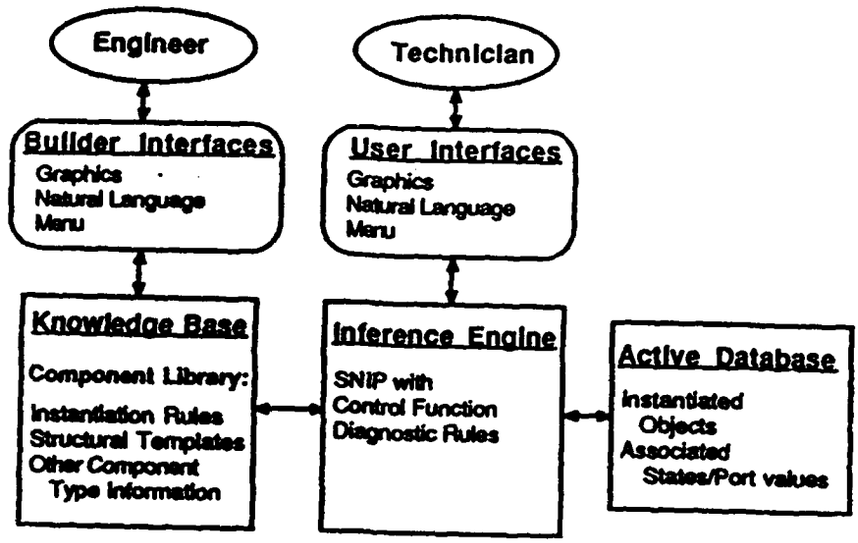


Figure 1 Architecture of VMES.

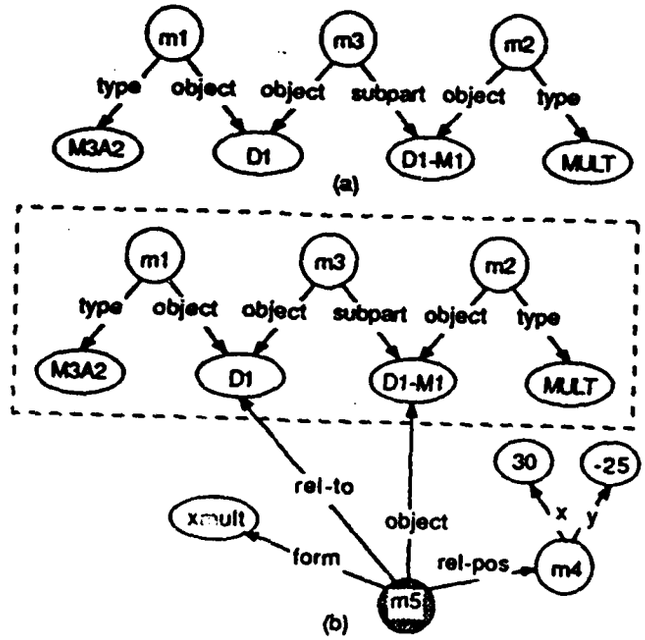


Figure 2 A unified knowledge representation using SNePS. (a) The SNePS network representing "D1 is an M3A2, D1-M1 is a MULTiplier, and D1-M1 is a subpart of D1". (b) Adding graphical knowledge without changing the original representation in (a).

User interaction is an important issue of the VMES project in two aspects: VMES has to communicate with the maintenance technician for test and repair, and it has to provide an engineer or a senior technician facilities for adapting it to other devices by adding their descriptions to the component library. In the next section, the VMES device representation scheme is described with the emphasis on device representation which facilitates a better user interaction. Section 3 discusses the knowledge-based graphics package of VMES, which maintains and reasons on "graphical deep knowledge" when interacting with the user. Section 4 is the conclusion.

DEVICE REPRESENTATION

In VMES, a device is modeled as hierarchically arranged modules. While this is hardly a new idea, the innovative part of our work includes: the use of an expandable component library; a clear distinction between two levels of abstraction of an object; representing a component "type" as an instantiation rule and a structural template; an explicit representation of wires and points of contact (POCONs); and the incorporation of logical and physical structure of devices for both diagnostic reasoning and user interaction.

General Representation Scheme

Devices in the digital circuit domain share many common component types. Representing every detail of a device causes much representation overhead, which in turn leads to system inefficiency. Instead of coding each device, we only describe the component types used by the device to the component library. Parts of a device are instantiated as needed. Adapting VMES to a new device is an easy task — just adding to the component library those component types used by the new device and not already in the component library. Since the representation scheme is still being experimented with, we currently have only about twenty different component types in the component library.

A component type is abstracted at two levels. At level-1, it is a module (black box) with I/O ports and a functional description. At level-2, its subparts and connections are described. In a previous implementation, these two levels were represented as two instantiation rules [7]. Since, usually, only a few subparts of an object are relevant to further diagnostic investigation, instantiating all subparts is inefficient. As an improvement, the two levels are now represented as an instantiation rule and a structural template [10]. An instantiation rule instantiates an object with its ports and functional associations. A structural template is a piece of passive knowledge, which allows VMES to search the suspicious subparts of an object. Unlike other procedural representations of the level-2 abstraction [1, 7], the structural template itself is never fired or copied. Since the investigation of a suspect is often terminated without checking its subparts, the clear distinction of the two levels of abstraction along with their separate representation has advantages on diagnosis and representation efficiencies.

Explicit representation of wires and POCONs is necessary for diagnosing faults of circuit connections [11]. The traditional model of a wire as a unidirectional module is inappropriate, because it ignores its bi-directional nature, and it does not include POCONs. In VMES, a wire is modeled as a bi-directional module to preserve its physical property, and its uni-directional design intention is retained by the connection mechanism. Components are connected either by forming a POCON from two different ports or by superimposing two ports,

which are a same port abstracted at two different hierarchical levels, together. With this new model, VMES is able to locate interrupted wires and bad contact points.

Adding Physical Representation

Human diagnosticians of electronic devices seem to simultaneously maintain models of the logical and physical structures of the target device. They carry out most of the diagnostic reasoning over the logical structure of the device due to its functional association. While carrying out the reasoning, the logical structure is apparently mapped to the physical structure from time to time. Tests and measuremen :s are first initialized using the logical structure, and then are realized and executed on the physical structure. Repair, which is usually done by replacing a physical unit or by fixing a physical connection, is planned and done on the physical structure. In other words, maintenance technicians use a model of physical structure of the target device, which is a hierarchically arranged set of replaceable physical components at various maintenance levels such as field-level and depot-level. By mapping the logical structure of the device to its physical equivalent, maintenance technicians are able to terminate the diagnostic process at the right moment and to form an adequate repair plan.

Given that the mapping between the logical structure of the device and its physical equivalent happens throughout the diagnostic process at all hierarchical levels, the speed in carrying out the mapping is critical to the time needed to locate faults. This implies that objects on both the logical structure and the physical structure of the device should be closely linked to each other so that the mapping is done efficiently. Even experienced technicians may have difficulty in locating a point of a schematic diagram on the real device, where the schematic diagram represents the logical structure of the device, and the form of the real device is the physical structure; which is attributable to the large difference between the logical and the physical structures and a lack of cross-links at all hierarchical levels of the device in human memory. On the other hand, when modeling and representing a device in an automatic fault diagnosis system, the cross-links between its logical structure and physical structure can be modeled and represented to an appropriate level of detail. This is indeed possible to do in a computer with reasonably sized memory.

In VMES, the physical structure of a device is represented distinctly from but in a similar way as its logical structure. In a structural template for a logical component type, every subpart of the component type is specified with a subpart "id" and a subpart "type", which are used to instantiate the subpart if it is found to be a suspect and further investigation of it is necessary. In addition to the subpart "id" and "type", an "mntn-lv" indicator is also associated with every subpart of a physical component type. The "mntn-lv" indicator shows the intended maintenance level of the subpart, i.e., the maintenance level where the subpart, if found faulty, is replaced without further diagnosis. The "mntn-lv" indicator is associated with the physical structure rather than the logical structure of a device to reflect the fact that human experts form and carry out a repair plan based on a physical model rather than a logical model of the device.

In order to abstract a device into a model, which can be efficiently represented and interpreted, some abstraction restrictions have to be made. First, the hierarchical trees abstracted from the two perspectives should have the same number of hierarchical levels. Second, the cross-links can only be made at the same hierarchical level. Third, several logical objects on the logical structure can correspond to the same physical object on the physical tree, but a logical object

can not spread over several physical objects. This restriction seems unreasonable at first, but a closer investigation of the electronic domain shows the contrary — physical objects in the domain usually have larger grain size than logical objects. This is especially true with modern technology as more and more logical functioning units are being packed into a physical unit, e.g., a simple HexInverter chip (a physical object) contains six independent inverters (logical objects).

Cross-links between Representations

The two representations of the logical and the physical structures of a device are cross-linked at every hierarchical levels. There are two kinds of cross-links between the logical structure and the physical structure of a device. The first kind are cross-links for components. The second kind are cross-links for ports. Cross-links for components are implemented by the "object<*logicalObj*>/ inside<*physicalObj*>" semantic network case-frame (Fig. 3(a)). No distinction is necessary as to whether the physical object contains a single logical object or several logical objects. This is because we just care about whether the corresponding physical object of a faulty logical object is at the intended maintenance level and should be replaced, or it is not and the diagnostic process should continue; this is independent of whether the physical objects contains anything else. (Actually, a physical object in the electronic domain is often replaced with most of its parts being intact.)

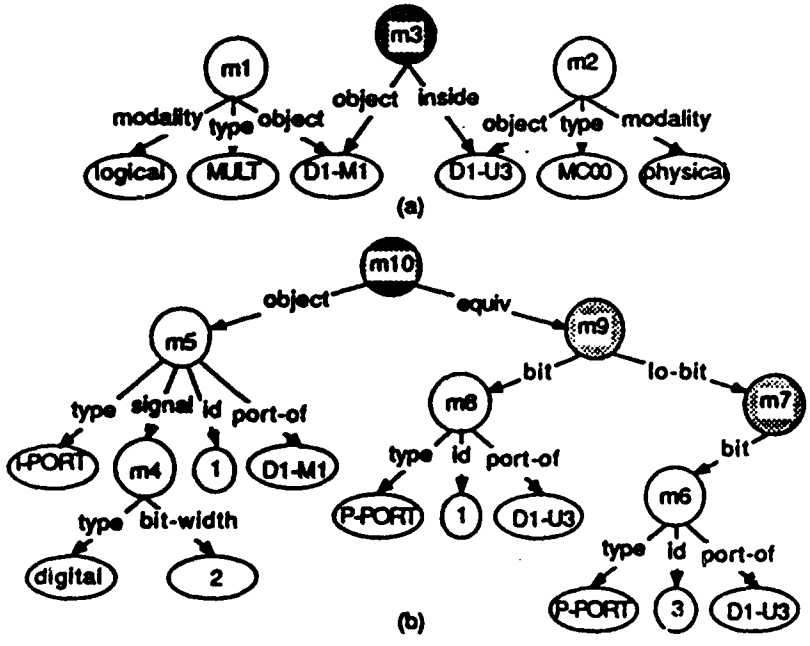


Figure 3 Representation of cross-links between the logical and the physical structures of a device.
(a) Component cross-links. (b) Port cross-links.

While the cross-links of components helps in determining if the diagnostic process should go on or terminate, and in forming a repair plan, the cross-links of ports makes user interaction much easier — when ordering a test or a measurement, it can be used to clearly direct the user to the right location on the real device. It is implemented by the "object<*logical.port*>/equiv<*physical.port*>"

semantic network case-frame (Fig. 3(b)). The advantage of a logical abstraction of the device is that it provides a high level view of the device which facilitates the diagnostic reasoning. For instance, a n-bit wire is abstracted as a single logical wire, thus freeing the technician (or a fault diagnosis system) from thinking about bit slices. However, when a measurement is required, it is necessary to locate all the bit-ports on the real device, and this is often a difficult task since these bit-ports may spread out randomly. In our representation (Fig. 3(b)), these bit-ports (physical ports) are linked together, from high-order bit to low-order bit, by the recurrent case-frame of "bit<*a.physical.port*>/ lo.bit<*the.remaining.lower.bits*>".

Physical Representation at Work

In the rest of this section, we describe how VMES uses this device representation to facilitate fault diagnosis and user interaction. When bad outputs are found in the suspect currently being investigated, the system has to determine if the diagnosis should terminate or not. Most fault diagnosis systems use the simple idea of SRU (smallest replaceable unit) which says that the diagnostic process stops when the current suspect is a SRU, i.e., a terminal node (a leaf) of the structural hierarchical tree of the device [1,2]. VMES takes a more flexible approach by incorporating the idea of "intended maintenance level" into the system. A system parameter, VMES.IML, is set to the "intended maintenance level" the system is working on. If a part shows some bad outputs and it is at the intended maintenance level, it is declared faulty and the diagnosis on it is terminated. For example, a board is replaced at *field* and then sent back to a *depot* where the fault is further isolated to a chip. The checking for the maintenance level of a part is done on the corresponding physical object of the part (a logical object), and a repair plan is formed based on the component type of the physical object. VMES also provides an opportunity for the user to short-cut the diagnosis by noticing that all remaining (logical) suspects are in a single replaceable physical unit at VMES.IML. Since the same physical object gets replaced no matter which logical suspect is faulty, further discrimination among the suspects are unnecessary provided that connections are assumed to be intact.

The major interaction between VMES and the user is the input of port values. Since diagnostic reasoning is carried out on the logical model of the device, VMES always wants the value of a logical port. Through the cross-links between logical and the physical structures, VMES is able to inform the user which "physical ports" should be measured for a logical port. Note that in digital circuits, a logical port may corresponding to several randomly spread-out physical ports (or pins of chips). Two examples of how the physical representation of a device helps the user in executing a port value measurement are shown in Fig. 4. The port to be measured in Fig. 4(a) is a port of a common component (non-wire component); and the one in Fig. 4(b) is a bi-directional port (a wire-end) of a wire. For representation and display efficiencies, wires are excluded from the physical representation of a device; this does not hurt the user interaction since the wire-end of a wire can always be identified as the wire-end connected to a port of a common component in the physical representation as shown in Fig. 4(b). Note that two kinds of values a user can type in: decimal and binary, where the binary numbers are prefixed by the letters "B" or "b". The user interaction shown in Fig. 4 is through pure text in SNePSUL (SNePS User Language) format, it can be improved by implementing it in natural language and graphics [8].

```
what's the value of port
(m316 (signal (m152 (bit-width (4)) (type (D))))
        (id (inp2)) (port-of (D1-A2)) (type (I-PORT)))
Equivalent Physical Port from Hi-bit to Lo-bit:
(m398 (id (2)) (type (P-PORT)) (port-of (D1-U2)))
(m399 (id (4)) (type (P-PORT)) (port-of (D1-U2)))
(m400 (id (6)) (type (P-PORT)) (port-of (D1-U2)))
(m401 (id (8)) (type (P-PORT)) (port-of (D1-U2)))

* [value]/nil? B0110
```

(a)

```
what's the value of port
(m291 (signal (m35 (bit-width (2)) (type (D))))
        (id (1)) (port-of (D1-W2)) (type (B-PORT)))
Equivalent Physical Port from Hi-bit to Lo-bit:
The WIRE-ENDs connected to
(m427 (id (6)) (type (P-PORT)) (port-of (D1)))
(m428 (id (7)) (type (P-PORT)) (port-of (D1)))

* [value]/nil? 2
```

(b)

Figure 4 Asking a port value measurement. (a) On a common component. (b) On a wire.

```
>>>>> I GOT THE FAULTY PARTS AS >>>>>
(D1-M2)
$$ Repair Order: replace D1-U3 (type:MC00)
done
```

(a)

```
>>>>> I GOT THE FAULTY PARTS AS >>>>>
(D1-W1)
$$ Repair Order: fix the wire connecting
(m420 (id (4)) (type (P-PORT)) (port-of (D1)))
(m389 (id (8)) (type (P-PORT)) (port-of (D1-U3)))
(m429 (id (4)) (type (P-PORT)) (port-of (D1-U3)))
$     and also the wire connecting
(m419 (id (3)) (type (P-PORT)) (port-of (D1)))
(m388 (id (10)) (type (P-PORT)) (port-of (D1-U3)))
(m428 (id (2)) (type (P-PORT)) (port of (D1-U3)))
done
```

(b)

```
>>>>> I GOT THE FAULTY PARTS AS >>>>>
(m324 (contact (m322 (signal (m35 (bit-width (2)) (type (D))))
                        (id (3)) (port-of (D1-W1)) (type (B-PORT)))
               (m323 (signal (m35 (bit-width (2)) (type (D))))
                        (id (inp1)) (port-of (D1-M2)) (type (I-PORT)))))
$$ Repair Order: fix the contact point at
(m388 (id (10)) (type (P-PORT)) (port-of (D1-U3)))
done
```

(c)

Figure 5 Repair suggestion made by VMES. (a) On a common component. (b) On a wire. (c) On a POCON.

The third use of the physical representation of a device is in repair suggestions. When a faulty object is found or at the end of the diagnosis session, VMES suggests a repair plan to the user according to the type of the faulty object (Fig. 5). If the faulty object is a common component, VMES just suggests that the user replace its corresponding physical part. If it is a wire, the corresponding physical wires are identified for repair. Note that a logical wire may correspond to several physical wires, for example, a 4-bit logical wire is realized by four wires on a printed circuit board; only the physical wires which are responsible for the fault are identified for repair. This is done by decomposing the port value of a logical wire into bit slices to determine which bit(s) are giving incorrect values. Finally, if the faulty object is a POCON (point of contact [11]), that is, it is a bad contact point, the user is directed to the location of the contact point. The physical representation is not only used to form the repair plan, it also helps direct the user to the object or the location on the real device where the repair is actually performed. In other words, it provides for better user interaction in both test and repair.

THE INTELLIGENT USER INTERFACE

General Remarks

As described in [7] VMES contains a knowledge based graphics package which is used as part of the VMES user interface. The purpose of this part of the VMES project is to investigate new designs for user interfaces, and to investigate what we have called "Graphical Deep Knowledge". We consider a knowledge representation system to be dealing with Graphical Deep Knowledge (as opposed to graphical knowledge), if the knowledge is organized in a way that makes it accessible not only to display routines, but also supports some form of graphical reasoning with this knowledge.

Naturally, a procedural knowledge paradigm is not acceptable for Graphical Deep Knowledge. While many graphics systems eliminate all information not essential to the purpose of display, our system contains *prima-facie* "redundant" information that is not immediately necessary for display purposes. However, we have found good reason to maintain this additional knowledge and have at least four reasons why additional knowledge adds to the power of a representational system.

(1) It is helpful for a system to know what is currently visible on the screen. A graphical representation looses much of its power if the user cannot refer to the objects shown by that representation. One can convince oneself easily of the importance of this notion by looking at virtually any system of graphical representation (including the diagrams in this paper). The given figures are always referred to by some text and derive their explanatory power from this interaction with the text. However, this requires that the system think in the same relations as the user and maintain the same conceptual units as he does.

(2) Declarative representations are required for any logic based reasoning. This factor has been the prime motivation for the representational tools developed here. An example of a simple reasoning operation would be a situation where the position of one object O1 is known, and it is also known that this object has an indeterminate spatial relation to another object O2, like e. g. leftness. Any person could immediately derive from these facts that the object O2 must therefore be somewhere to the right of O1, and any system that could not follow this step of reasoning would be

considered by a user as not intelligent. Reasoning in the domain of graphics is especially interesting, because not only traditional forms of logic based reasoning have to be investigated, but also analog reasoning is of interest.

(3) Modern software engineering has made the concept of modularization mandatory, and very often production programmers divide a program into modules, such that the user interface is one module, and the *host program* that performs the services the user is really interested in is another module. This leads to the existence of an interface between host program and user interface which, according to methods of good program design, has to be kept well defined and small. The result of this modularization is that the important concepts of the host program are not available to the user interface, and vice versa. Therefore a user can only refer to units the system designer decided explicitly to export from the host program. Knowledge based programming permits the sharing of information between different modules without creating a bottle-neck between them, and without having dangerous global information available to several modules. The reason why a knowledge base is not anywhere near as dangerous as the sharing of global variables is that knowledge bases are dealing with facts that are considered generally true, and if a fact is true there is no reason to keep it private to a single module, and little danger of conflicting definition or access. This last statement is especially true for rule-based systems. A rule expressing that A is left of B if and only if B is right of A is a universal truth that can be made available to any module in any system.

(4) If a knowledge based system supplies tools for natural language interaction a knowledge base containing Graphical Deep Knowledge becomes in an interesting sense an interlingua, namely an interlingua between the visual and the linguistic faculties of the system. Given that most knowledge based systems have been created with some consideration of natural language processing this observation should be of general interest to KR research. Specifically the SNePS system has a number of tools for natural language processing which permit the implied interactions.

Some other comments on the use of knowledge based methodologies in user interface design can be found in [4] and [9].

The TINA Graphics Interface to VMES

Three user interfaces have been developed for the VMES system which are in different states of completion and integration with the maintenance reasoning program. The first interface is the "TINA" program for knowledge based image generation. This program maps a declarative knowledge structure into a diagrammatic representation on a visual display device. This module exists in two versions with different focus, one of which has been used in the past by the maintenance reasoner to inform the user about the current state of the diagnosis process. Details of this representation have been reported elsewhere [7], but it should be pointed out that symbol colors are an important aspect of this representational facility.

It is relevant to the discussion of TINA that an important mode of display for which the theoretical ground work has been layed in this project, called the Intelligent Machine Drafting mode, has been developed. Traditional CAD systems for circuit boards are usually concerned with the maintenance of graphical representations of wire plans that show a physical picture of the device. In

contrast, technicians usually look (also) at logical wire plans. Logical wire plans are interestingly different from physical ones in that no absolute positions for components need to be maintained, only certain connectivity relations.

Nevertheless it is possible to draw the same wire plan in two different ways such that one of them brings across the idea of the representation, and the other one doesn't. It is the goal of TINA in IMD mode to create a logical diagram that is "easy to read", by laying it out and routing it not according to principles of CAD like minimization of energy consumption, but according to principles of minimal cognitive complexity. The most important such principle that has been used is the *equal distribution* of structure in the given space.

It goes without saying that a knowledge base for use with IMD mode does not contain any knowledge of coordinates, and that the major effort in the process of display is the reconstruction of this knowledge from hierarchy and connectivity information. The abilities of the IMD module in use are limited to a small device class that we refer to as A*M* and which has been modeled around the "Adder-Multiplier" (Fig. 6), a device famous in the maintenance literature. A*M* permits small variations of the Adder-Multiplier, for instance variations in the number of ports per components, in the number of components per row, in the number of processing rows, and in the number of connections per port. A formal description of the device class has to be omitted due to limitations of space.

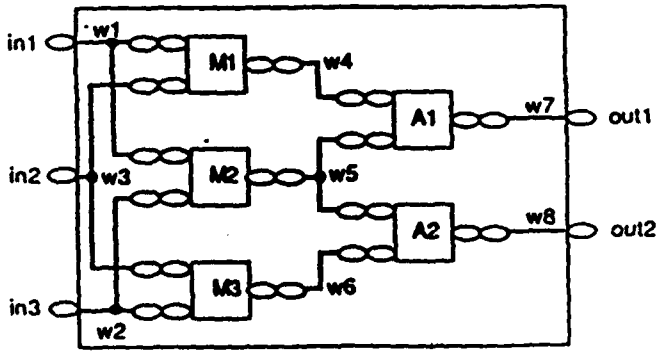


Figure 6 A 3-multiplier/2-adder board.

The Readform Interface for Object Creation

The second interface is the "Readform" program which is used for the creation of visual icons in a format that is accessible to the knowledge representation system. This avoids the necessity of hand generation of graphics code. The compilation of a larger pictorial unit is done by asserting information about objects in the network, such that in the process of drawing access is made to the icons created by Readform. A knowledge based version of Readform has been in the process of development for some time, however as of this writing only the theory of this system will be claimed.

By observing users in the process of object creation (with Vanilla flavored Readform) it has become obvious that the internal conceptual structures of the person can to a certain degree be derived from the order of his actions as well as by asking a few questions at strategic points. Readform supplies the user with a scratch buffer which is separate from the object created at the current moment.

Users have been observed to create objects by drawing a simple unit in the scratch buffer and then repeatedly yanking the buffer content into the picture.

From this chain of actions one can derive that all the yanked objects are presumably members of a certain class, and the system can verify this by asking the user whether there is in fact such a class, and if so, how to name it. This information can be used to create exactly the Graphical Deep Knowledge Structures that have been mentioned before as being used for picture generation.

The other thing that can be derived from the above chain of user interactions is that all the yanked icons are presumably parts of a larger object which consists of all the iconic primitives (lines, arcs, etc.) which were not created by using the scratch buffer. This permits a system to ask the user whether he wishes to name this larger object separately, and if he desires so, a part relation between the yanked parts and the main structure can be formulated and stored in the knowledge base as a proposition. This proposition becomes part of the part hierarchy in the knowledge base. Part hierarchies are a backbone of many representational systems and are used in the process of maintenance reasoning as well as having major importance in the derivation of pictures from Graphical Deep Knowledge and in controling complexity of displayable pictures.

The third user interface that we will treat in this paper is the natural language interface.

The Natural Language Interface

A versatile maintenance system is in need of a user interface in two different situations. In the first situation a maintenance technician uses the system to get help in troubleshooting a currently faulty device. The second situation is as important, namely the initial creation of the device representation. In order to deserve the title "versatile" it must be possible to create device representations with ease and flexibility. The natural language interface that will be described here belongs to the second class of interfaces. It is the goal of this interface to create an internal device representation to the point where it is possible to display the whole device. However, as much of this creation as possible should be done with natural language.

As has been pointed out in the section on Intelligent Machine Drafting, there is no necessity to actually enter coordinate information, so the natural language descriptions become quite natural. Natural language processing is done by way of an ATN interpreter/compiler that is part of the SNePS environment [6]. The class of objects that can be built by natural language is limited, even in comparison to the already limited class A*M* of displayable devices. The major additional limitation that is imposed by the language interface is the branching factor of electrical connections. It is possible to create wires impinging on at most three port.

Below, the original set of sentences that is understood by the NL interface and that describes the Adder-Multiplier will be presented. Running this set of sentences through the ATN interpreter will create all the structures necessary to describe the Adder-Multiplier *completely* for display purposes.

(nl)
D1 is a board
D1M1 is a multiplier
D1M2 is a multiplier
D1M3 is a multiplier

D1A1 is an adder
D1A2 is an adder
D1 has 3 inports
D1 has 2 outports
D1M1 has 2 inports
D1M1 has 1 outport
D1M2 has 2 inports
D1M2 has 1 outport
D1M3 has 2 inports
D1M3 has 1 outport
D1A1 has 2 inports
D1A1 has 1 outport
D1A2 has 2 inports
D1A2 has 1 outport
connect input 1 of D1 with input 1 of D1M1 and input 1 of D1M2
connect input 2 of D1 with input 2 of D1M1 and input 1 of D1M3
connect input 3 of D1 with input 2 of D1M2 and input 2 of D1M3
connect output 1 of D1M1 with input 1 of D1A1
connect output 1 of D1M2 with input 2 of D1A1 and input 1 of D1A2
connect output 1 of D1M3 with input 2 of D1A2
connect output 1 of D1A1 with output 1 of D1
connect output 1 of D1A2 with output 2 of D1
D1M1, D1M2, D1M3, D1A1, and D1A2 are parts of D1
wires are parts of D1
the form of a board is xboard2
the form of a multiplier is xmult2
the form of an adder is xadd2
the form of a PORT is xport
end

The first (nil) above calls the natural language processor from the SNePS environment, while the "end at the end returns to the SNePS environment. Although the vocabulary of this interface is quite limited there are variations of the sentences shown above possible.

Of special interest are the final sentences that start with "the form" because these sentences call, if necessary, the before mentioned Readform interface from inside the ATN interpreter and not only assert the relations between object class and form, but also create any unknown form-icons by having the user draw this icon. If the form is already known to the system, then only the assertional component of this operation will be executed.

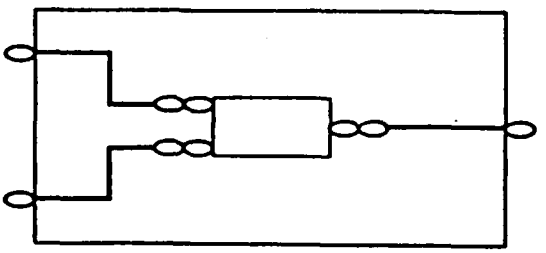


Figure 7 A low-end member of M*A*.

This last operation involves the call of Readform from the ATN interpreter which itself runs embedded in SNePS which itself runs on top of Franz LISP. While display operations are extremely slow (hours on a VAX 11/750 for non-trivial layout tasks), the natural language interface works reasonably fast (response times under a minute) considering especially the multiple layering of the used systems.

A device consisting of a single multiplier with two inputs and one output and the three wires needed to connect the multiplier to the three device ports, plus the six biports of the wires themselves are laid out in about 20 minutes (Fig. 7). This device has also been created completely by natural language interactions and it represents a low end member of the A*M* class.

CONCLUSION

In diagnostic problem solving, human experts seem to use both the logical structure and the physical structure of the target device throughout the diagnostic process at every hierarchical level. Knowledge of the logical structure of the target device together with the associating functional knowledge is used for diagnostic reasoning, and knowledge of its physical structure is used to carry out a test, to determine when the diagnostic process be terminated, and to form a repair plan. It is important to incorporate the physical representation and the logical representation of a device in maintenance. We find that a physical representation of the target device, together with the representation of the cross-links between the logical and the physical structures of the device, contributes to fault diagnosis in several aspects — such a system does not merely mimic the behavior of human experts, it may outperform human experts in certain situations. It helps determine when a diagnostic process should be terminated, thus it provides versatility across maintenance levels. It can provide a short-cut to diagnosis by noticing that all logical suspects are in a physical object at the intended maintenance level. It helps to form a repair plan based on the physical nature of the target device. Finally, (probably the most important point,) physical representation eases user interaction: it helps direct the user to the exact location in the real device for test and repair.

It has been argued in this paper that knowledge based methodologies are of increasing importance for intelligent systems. They permit intelligent behavior of the interface and help to offset problems in information privacy that are enforced by modern software engineering technology. A powerful set of user interfaces is also a precondition for a versatile system, because most expert systems have to talk with people of different requirements during their life cycle. Specifically three user interfaces have been introduced in this paper. The first one caters to the end user, which for VMES is the maintenance technician. It creates graphical representations of circuit boards. The specific research contribution of this part of VMES is the creation of logical wire plans without any prior knowledge about coordinate values of the system icons. The other two interfaces are mainly of use for the device designer who wants to enter information about a newly created device into the maintenance system, without having to learn some obscure graphics or KR language. The first of these two interfaces permits the creation of graphical icons of new components. This interface is called from inside the natural language interface, if a user attempts to use a primitive form which was not previously declared. The natural language interface is based on the SNePS ATN interpreter, and the complete necessary natural language input for the creation of an artificial device called the Adder-Multiplier has been

presented.

ACKNOWLEDGEMENT

This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB DC 20332 under Contract No. F30602-85-C-0008, which supports the Northeast Artificial Intelligence Consortium (NAIC).

REFERENCES

1. R. Davis, "Diagnostic Reasoning Based on Structure and Behavior," *Artificial Intelligence 24* (1984), 347-410.
2. M. R. Genesereth, "The Use of Design Descriptions in Automated Diagnosis," *Artificial Intelligence 24* (1984), 411-436.
3. D. P. McKay and S. C. Shapiro, "Using Active Connection Graphs for Reasoning with Recursive Rules," in *Proc. of IJCAI-81*, William Kaufman, Los Altos, CA, 1981, 368-374.
4. B. Neches and T. Kaczmarek, "Knowledge Based Interfaces," in *AAAI-86 Workshop on Intelligence in Interfaces*, August 14, 1986.
5. S. C. Shapiro, "The SNePS Semantic Network Processing System," in *Associative Networks: The Representation and Use of Knowledge by Computers*, N. V. Findler (editor), Academic Press, New York, 1979, 179-203.
6. S. C. Shapiro, "Generalized Augmented Transition Network Grammars For Generation From Semantic Networks," *The American Journal of Computational Linguistics 8*, 1 (1982), 12-25.
7. S. C. Shapiro, S. N. Srihari, M. R. Taie and J. Geller, "VMES: A Network-Based Versatile Maintenance Expert System," in *Proc. of 1st International Conference on Applications of AI to Engineering Problems*, Springer-Verlag, New York, April 1986, 925-936.
8. S. C. Shapiro and J. Geller, "Artificial Intelligence and Automated Design," in *Proc. of the SUNY Buffalo Symposium on CAD: The Computability of Design*, SUNY at Buffalo, NY, 1986.
9. S. C. Shapiro and J. Geller, "Knowledge Based Interfaces," in *AAAI-86 Workshop on Intelligence in Interfaces*, B. Neches and T. Kaczmarek (editor), August 14, 1986, 31-36.
10. M. R. Taie, S. N. Srihari, J. Geller and S. C. Shapiro, "Device Representation Using Instantiation Rules and Structural Templates," in *Proc. of Canadian AI Conference - 86*, Presses de l'Université du Québec, Montréal, Canada, May 1986, 124-128.
11. M. R. Taie and S. N. Srihari, "Modeling Connections for Circuit Diagnosis," in *Proc. of The 3rd IEEE Conference on AI Applications*, IEEE Computer Society Press, Orlando, FL, Feb. 1987, 81-86.

Appendix A5:

"Graphical Deep Knowledge for Intelligent Machine Drafting"

Geller, J.
Shapiro, S.C.

GRAPHICAL DEEP KNOWLEDGE FOR INTELLIGENT MACHINE DRAFTING

James Geller and Stuart C. Shapiro
Department of Computer Science
State University of New York at Buffalo
Buffalo, NY 14260
geller%buffalo.bitnet@cunyvm.cuny.edu

ABSTRACT*

The problem of Intelligent Machine Drafting is presented, and a description of an existing implementation as part of a graphical generator function is given. The concept of Graphical Deep Knowledge is defined as a representational basis for Intelligent Machine Drafting problems as well as for physical object displays. A (partial) task domain analysis for Graphical Deep Knowledge is presented. Primitives that are necessary to deal with a world of 2-D forms and colors are introduced. Among them are primitives for describing forms, positions, parts, attributes, sub-assemblies, and an abstraction hierarchy. The use of the "linearity principle" for knowledge structure derivation from natural language utterances is shown.

I INTRODUCTION

Traditional computer graphics systems have been criticized in the literature for being a poor environment from a knowledge representation point of view [1]. In graphics as well as in other areas of software development, programmers have been taught not to code any items that are irrelevant to the actual execution of a given task. Over the last several years it has been an essential goal of AI programming to reverse the process of elimination of knowledge from the coding process. The AI programmer tries to make his knowledge conscious and tries to incorporate much of it in his program.

The use of AI techniques in other areas of computer science has led to the replacement of the "eliminate knowledge" paradigm by the "add knowledge paradigm" outside of AI proper. In this sense we interpret Brown et al., and in this sense we want our work to be understood.

In the setting of the VMES project (Versatile Maintenance Expert System) for printed wiring board maintenance [2] we have been working on a knowledge based graphics system. In this paper a new class of layout/routing problems that we have encountered will be described (Intelligent Machine Drafting), and a task domain analysis of what we call Graphical Deep Knowledge will be given.

A. The Display Program

A major part of the VMES user interface is a display program (named TINA), which is called by the maintenance assistant of VMES and keeps the user constantly informed what VMES is currently "thinking" about. For this purpose it displays, using certain symbol colors, a logical diagram of the circuit board currently being analyzed. Suspected components are displayed in green. Components found faulty are displayed in red. Violated expectations are shown in magenta. Objects about which nothing negative is (yet) known are displayed in blue. Blinking is used to indicate the current focus object of the system.

VMES is implemented on top of SNePS the "Semantic Network Processing System" [3]. The display program is conceived of as a generator that creates pictures from a knowledge base. This is in total analogy to the generation of natural language from a knowledge base.

B. The Linearity Principle of KR

One preliminary idea that has been guiding our work is what we call the *linearity principle*. Let there be two systems S_1 and S_2 consisting both of a parser (P_1, P_2) and a knowledge representation formalism (K_1, K_2). Let there also be a function O_1 that can be applied to any natural language expression and a function O_2 that can be applied to any knowledge structure of K_1, K_2. O_1 and O_2 compute some unspecified complexity measure of their arguments.

Both systems S_1 and S_2 impose a mapping from a natural language input to a knowledge structure. We will call these two mappings M_1 and M_2 respectively. We now compute two functions f_1, f_2 such that

$$f_1(u) = \frac{O_2(M_1(u))}{O_1(u)}$$

$$f_2(u) = \frac{O_2(M_2(u))}{O_1(u)}$$

In the above formulas "u" describes a syntactically valid and semantically meaningful natural language utterance contained in the domains of both P_1 and P_2. We will call the system S_1 better than the system S_2 if $f_1(u)$ can be approximated better by a constant than $f_2(u)$. A practical judgement about the constancy of f_1 and f_2 could be done by computing mean and standard deviation of f_1 and f_2 for a large number of different u values, however we will limit ourselves to intuitive judgements.

The Linearity Principle:

> The quotient of the complexity of a knowledge structure and the natural language utterance that it represents should be approximately a constant for any given parser and KR system.

Intuitively the linearity principle says that we do not want to represent most five word sentences of a language with three or four semantic network nodes, but have one five word sentence of this language represented with 25 nodes. An implicit application of the linearity principle (LP) can be seen in Shapiro's work on non-standard connectives [4].

Before we present an example application of the LP it is necessary to say that the arc labels in SNePS networks (Fig. 1, more explanations will be given in the next section) are seen as system primitives. The number of different arc labels is not fixed and can be expanded by the user. It is assumed that the number

* This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB DC 20332 under Contract No. F30602-85-C-0008, which supports the Northeast Artificial Intelligence Consortium (NAIC).

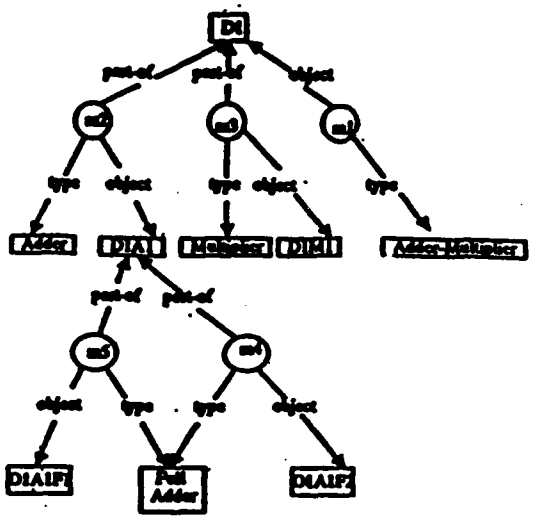


Figure 1

A SNePS network giving (partial) information about an Adder-Multiplier

of arc labels necessary for one limited domain will converge towards a stable set, therefore identifying this set is a way of task domain analysis [5].

If people can describe a simple arrangement of objects by a short sentence then it should be possible to describe it with a reasonably simple SNePS structure. If this is not the case then the number of user defined primitives has to be extended to accommodate the sentence. (Of course new primitives will also have to be used if the sentence is not representable at all).

For instance if two people are sitting in front of a graphics terminal displaying the Adder-Multiplier (which has been used in maintenance research, Fig. 2), and one of them asks:

"Tell me the names of all multipliers."

then the other person will presumably be able to do that. Therefore we would want our graphics interface to be able to do the same thing. We also want the knowledge base to contain information on all multipliers in a format approximately linear in size with respect to the answer given by a person. This leads directly to an old idea, the implementation of a class hierarchy. (Less obvious examples will be given throughout this paper.)

C. Notational Conventions for SNePS networks

Fig. 1 shows an example of a typical SNePS network in order to provide some intuition for the reader not familiar with SNePS. The syntax and semantics of SNePS have been carefully defined [6]. SNePS is also a "neat" KR system that incorporates full first order predicate calculus. We will use Fig. 1 to introduce the network notation that will be used in this paper.

The nodes m1, m2, m3, m4, m5 represent propositions. m2 expresses the fact that the object D1A1 is of type Adder. An equivalent first order predicate calculus representation for Fig. 1 would be the following one:

type(m1,Adder-Multiplier) & object(m1,D1)
type(m2,Adder) & object(m2,D1A1) & part-of(m2,D1)
type(m3,Multiplier) & object(m3,D1M1) & part-of(m3,D1)
type(m4,Full-Adder) & object(m4,D1A1F2) & part-of(m4,D1A1)
type(m5,Full-Adder) & object(m5,D1A1F1) & part-of(m5,D1A1)

This representation is unnecessarily redundant, and we will introduce a pseudo-predicate notation according to the following formal scheme. Given a conjunction of a number of binary predicates with identical first arguments, transform the first argument into a pseudo predicate. Transform all binary predicates into arguments at odd numbered positions, and insert all second arguments at even numbered positions. In symbols:

$$\bigwedge_{i=1}^{n} p_i(a_0, a_i) \rightarrow a_0(p_1, a_1, p_2, a_2, \ldots)$$

This transformation is syntactic sugar and has no influence on the meaning of the representation which depends on the combination of system primitives (arcs). Therefore all the a_i's that will be given in the following sections are to be understood as examples.

II INTELLIGENT MACHINE DRAFTING

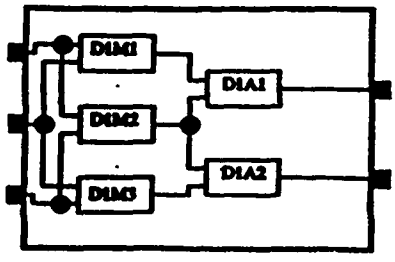
The creation of *logical* circuit board diagrams from a knowledge base is not addressed by a number of commercially available Computer Aided Drafting systems as well as research on CAD, layout systems, and routers [7,8]. Work has concentrated on layout and routing of physical diagrams. Logical diagrams are usually created with a graphics editor or by computer from hand sketches [9].

We are interested in layout and routing of logical circuit diagrams. Physical diagrams created by CAD systems have to be realized in hardware, and therefore the layout is usually optimized for signal length, area consumption, power consumption or heat dissipation. None of these requirements exist for logical diagrams. Rather one wants to create pictures that are optimized in a "human factors" sense [10]. The described difference can best be compared with the shift in attention in programming language research from space and time efficient programs to readable and maintainable languages.

Physical and logical routers also differ in their initial problem setting. A physical router prohibits wire crossings and makes use of different layers and "vias" to avoid them. A logical router permits wire crossings. It uses a special symbol (usually a dark dot at an intersection) to mark clearly whether a crossing is meant to be an electrical connection or not.

Def: Intelligent Machine Drafting (IMD).
Intelligent Machine Drafting is the activity of automatic creation of a cognitively appealing logical diagram of a system from a knowledge base which contains *no numerical coordinates* of the components of the system.

The application of IMD to circuit boards implies the need for a module that creates cognitively appealing layouts of all components and a logical (as opposed to physical) router that connects them.



D1

Figure 2

IMD has turned out to be an interesting problem from a theoretical point of view. It used to be the working method of engineering (and is still common) that a design engineer would send a hand sketch to a draftsman who would then draw a nicely laid out version of it. The job of the draftsman is usually considered a "low intelligence" position, requiring only a minor level of technical education. However this "low intelligence problem" differs from many "hard" AI problems because it cannot be translated easily into a symbolic representation. Solving IMD problems by humans requires use of (part of) the perceptual system, and possibly of the imagery system, both domains which researchers have not yet related well to the domain of problem solving.

A. IMD for circuit board display

We have defined a very limited class of objects called A*M* which is a simple generalization of the Adder-Multiplier of Fig. 2, and we have implemented a fully automatic layout and logical routing program for this class. A formal description of the class A*M* has been formulated. However, we will limit ourselves in this paper to an intuitive explanation.

- A device of class A*M* consists of at most one main object which is assumed to be a large box graphically containing all objects asserted as its parts. In the absence of this main object the screen itself is considered the main object.
- Signal flow in an object of the class A*M* as well as in its parts is strictly from left to right, with no feedback at any stage.
- The "signal length" (maximum number of components a signal has to pass through from system input to system output) and the "signal width" (maximum number of components that are active in parallel, assuming constant delay for every component) are small enough that linear chains of components can be constructed that will fit into the given main object, leaving enough additional space for wiring.
- The main object as well as all its parts have ports. Besides that there is no second level of the part hierarchy. (The ports of the main object are shown as little black boxes in Fig. 2.)

The current implementation makes a few additional assumptions which are not part of the A*M* definition, which however do not impose severe loss of generality and will be eliminated in the future. Among these assumptions are the constancy of the port size for all components and the assumption of small variation of size among components. Forward jumps of connections have not yet been implemented.

For efficiency reasons there is no backtracking programmed into the router, therefore one can design pathologically unsolvable cases rather easily, which does not currently concern us, given that the "general purpose routing problem" has not yet been solved either [7]. Work on IMD as well as on the display of *physical* board diagrams from a knowledge base have motivated our work on graphical deep knowledge (which will be defined in the next sections).

III GRAPHICAL DEEP KNOWLEDGE

A. Definition of Graphical Deep Knowledge

A large number of scientific fields make marginal to extensive statements about the representation of knowledge dealing with forms and colors. Space limits us to mention three main areas, natural language graphics [11], knowledge based graphics [12,13], and the imagery debate between "imagists" [14,15] and "propositionalists" [16,17]. The existence of some propositional representations is now widely accepted in all camps. However, many researchers seem to gloss over the details of their representation, just stating that problems up to a certain size could be done with a list of propositions. Kosslyn's implementation, a notable exception [14], implies an important criterion for a propositional representation of forms, namely that it can be used to create actual pictures. We want to call this criterion *projective adequacy*.

A system that also permits *reasoning* based on shapes or positions of objects will be said to demonstrate *deductive graphical adequacy*. The type of reasoning permitted is either analogical or propositional. In order to avoid any possible terminological confusions we will shun the terms "spatial knowledge", "visual knowledge" and even "graphical knowledge" and use the term "graphical deep knowledge".

Def: Graphical Deep Knowledge
A knowledge base is said to contain graphical deep knowledge if at least part of its knowledge exhibits deductive graphical adequacy, and part of its knowledge exhibits projective adequacy.

The term "deep" is used in analogy with deep structures in linguistics.

One major goal of our research is to create a base of graphical deep knowledge that is adequate for displaying and reasoning about objects in general and about the domain of circuit boards in particular. Our current analysis of graphical deep knowledge is given in the following sections.

B. Form Knowledge

Objects may have individual forms or inherited forms.

m1(object d1 form xand modality function) (1)

m2(object d2 type and-gate modality function) (2)
m3(sub-class and-gate class boolean modality function) (3)
m4(class boolean form xboolean modality function) (4)

(1) describes an object (individual) d1 that has a form xand. The last binary predicate "modality" is used to discriminate between different display modes. Circuit boards permit display of their wire plan (logical or functional representation) and of their physical structure. The forms used for these two displays are usually different, therefore the form proposition must be qualified by the display modality for which this item of knowledge is valid.

A form like "xand" is at the same time a node in the semantic net and a LISP function that, if executed, would draw a specific form. Form functions are parameterized by the starting position. Therefore one form function can display the same object at different positions, but no other modification is possible.

(2) assigns d2 to the class of and-gates which are by (3) recognized as a sub-class of the class of boolean components which by (4) are all assigned the same form, namely xboolean. We have never found it necessary to inherit a form using an intermediate class.

C. Position Specification

A large number of representations for positions is possible. All object positions refer to the position of an object's fixed reference point.

1. Concrete and Fuzzy Absolute Positions

m5(object d1 abspos m6(x 100 y 200) modality function) (5)

(5) describes an absolute position of d1. The position is given by the substructure m6 which contains actual coordinate values.

The pseudo predicate m6 has to be read as a structured individual, not as a proposition [6]. The implicit assumption of this representation is that coordinates are given in pixels of a graphics display device and are "relative to the screen", and therefore called absolute.

When people give a description of a picture, they typically do not use coordinate values but rather talk about objects in the center, at the top, or at the left of the screen. According to the linearity principle it is therefore necessary to represent these "fuzzy" absolute positions. (6) shows an example of a fuzzy absolute position.

```
m7( object d2 fabspos center modality function)          (6)
```

Currently the exact meaning of the fuzzy terms is still under investigation. We have done a psychological pilot study with 20 subjects to find out what people think "leftness" means, which has not yet been totally evaluated. Fuzzy absolute positions used in this experiment are top, bottom, left, right, center, upper left corner, upper right corner, lower left corner, and lower right corner. The term "fuzzy" is not related to Zadeh's fuzzy logic [18].

2. Relative Positions

Propositions about relative positions can be divided into different groups, according to a number of criteria. The first distinction is between numeric positions (what we refer to as "concrete" positions) and fuzzy positions. For numeric positions there are at least three different ways to interpret coordinate values. Values can be given in pixels, or they can be multiples of the sizes of either the object or the reference object involved.

The reference object might be given explicitly or implicitly. In the second case there must be a "super-part" of the object which will be used as the reference object. Finally it might be the case that a relative position is inherited from a class of objects. Many of the given representational possibilities can be combined with each other.

a. Fuzzy Relative Positions–As for fuzzy absolute positions the analysis of the semantics of fuzzy relative positions is still under way and based on experimental data.

```
m8( object d3 frelpos left rel-to d1 modality function)  (7)
```

(7) describes the proposition that d3 is left of d1.

Unfortunately there are a number of fuzzy relative position descriptions which do not rely on binary relations. A representation for "between", which has two reference objects is shown in (8). More difficult are "on-one-line", "together", and "forming-a-circle".

```
m9( object d99                                           (8)
    frelpos between
    rel-to1 d98
    rel-to2 d97
    modality function)
```

b. Concrete Relative Positions–We will begin this section with an example for a relative position measured in pixel coordinates. Fig. 3 shows as example a multiplier. The little black boxes in the picture are ports (sic!) of the multiplier and have their own forms.

```
m10( object port3                                        (9)
     relpos m11( x 24 y 4)
     rel-to D1M1
     modality function)
```

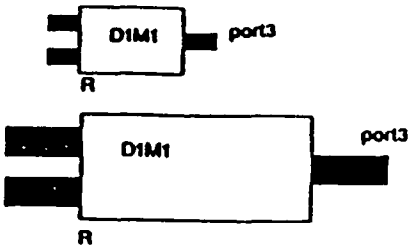


Figure 3
A Multiplier with 3 Ports
in 2 sizes. In both sizes
port3 is one bodylength away from R

(9) describes the relative position of port3 as being 24 to the right of D1M1, and 4 above it. Distances refer to the reference point R. If the relative position of a part of an object is given in pixel coordinates then a problem with scaling results: not only objects have to be scaled, but also relative positions. This is unsatisfying because it does not express the fundamental invariance of the position of the sub-part to its super-part. Fortunately it is possible to represent the relation between an object and its sub-parts preserving the conceptual positional invariance by using "body coordinates". These coordinates represent a relative position as multiples of the size of the relevant object.

```
m12( object port3                                        (10)
     relpos m13( bx 3 by 1)
     rel-to D1M1
     modality function)
```

(10) shows the same relative position as (9), however assuming that object port3 has a length of 8 pixels and a width of 4 pixels. The relative position "3", is a multiple of the size of port3. The length and width of an object are the length and width of the smallest surrounding rectangle of it which has lines parallel to the coordinate axes ("extent").

Intuitively, the representation expresses the fact that a big man has his arms far away from his neck, and a small child has its arms near to the neck, but the ratio of the distance and the size of the person should be approximately a constant.

Usually there will be a number of objects given with relative positions to the same reference object. This makes it desirable to specify relative positions in body coordinates of the reference object, shortly called reference object coordinates (denoted by the arcs brx and bry).

```
m14( object port3                                        (11)
     relpos m15( brx 1 bry 0.33)
     rel-to D1M1
     modality function)
```

(11) can be interpreted in the same way as (10), except that this time the factors (after "brx", "bry") apply to the size of the reference object, which is assumed to be 24 pixels long and 12 pixels high.

c. Explicit versus Implicit Reference Objects– In all cases so far the reference object of a relative position statement was given with a rel-to arc. In the circuit board maintenance domain a flat part hierarchy is used. There is one major object, the board, which has many different parts which should reasonably be placed relative to this main object. It would be redundant to assert the reference object for all the parts, and therefore a default assumption is practical.

m16(object d5 relpos m17(x 100 y -20) modality function) (12)

m18(object d6 sub-parts d5 modality function) (13)

(13) shows a part assertion, a descriptive tool that will be reviewed later on. Because of (13) the relative position asserted by (12) will be interpreted as being relative to d6.

Combinations of the representational constructs introduced are in general possible. For instance an implicit reference object may be used with all types of relative coordinates, including fuzzy ones (14, 15).

m19(object d7 f relpos left modality function) (14)

m20(object d8 sub-parts d7 modality function) (15)

d. Inherited Relative Positions–If one adder has its first port at half a body length from its reference point, then this will presumably hold true for all the adders in the system, and one would not want to assert this over and over again. A solution to this problem is to make the relative position itself inheritable. This option is exemplified by the following set of propositions. The relative position is given in reference object coordinates and inherited through an intermediate class "half-adder".

m21(object d9 type half-adder modality function) (16)

m22(object d10 sub-parts d9 modality function) (17)

m23(sub-class half-adder class adder modality function) (18)

m24(object d10 form rndd modality function) (19)

m25(class adder relpos m26(brx 2 bry .5) modality function) (20)

(16), (18), and (20) specify the relative position of d9 which is inherited from the class "adder"; (17) specifies the reference object d10 by force of its super-part relationship to d9. (19) is necessary to permit the derivation of the size of d10 which in turn is necessary for the computation of reference object coordinates.

3. Logical Reasoning with Fuzzy Positions

The following structure is a SNePS rule that expresses the fact that "if one object is left of another object, then the other object must be right of the first object and vice versa". For a detailed explanation of the structure of SNePS rules, see [19]

m27(avb (v1 v2 v3) (21)
  thresh 1
  arg (m28 object v1 rel-to v2 modality v3 f relpos left)
  arg (m29 object v2 rel-to v1 modality v3 f relpos right))

If the knowledge base contains the absolute position of B and the fuzzy position of B relative to A but no positional information about A itself, then A's fuzzy position can be derived with rule (21) or a variation of it.

D. Parts, Clusters, and Assemblies

Part hierarchies are a commonly used construct in AI [20]. Our research has indicated that a part hierarchy alone is not sufficient for graphical deep knowledge representations. We have added two other types of part-like hierarchies, called assemblies and clusters.

The display of a complicated object with several levels of parts might be impossible on a limited resolution display device. A natural way to limit the complexity of such a display task is to limit the number of levels of the hierarchy that are actually displayed. This is a very elegant solution because it does not require the introduction of any new representational construct.

(17) showed our representation of a simple part relation. An object can of course have more than one part. The ubiquitous modality attains a special importance for part hierarchies. Circuits like AND gates, OR gates etc. are displayed as single objects in a logical diagram. In real hardware there are usually four binary AND gates in a single chip. These four gates might be parts of different logical units. However, in a physical representation all four of them must be parts of the same integrated circuit.

1. Assemblies

Work on the maintenance part of the VMES project has led to the realization that certain objects should never be displayed without their parts. For instance, a port is a part of a multiplier, but a multiplier should never be displayed without its ports. Sub-assemblies are therefore objects that have a real part-whole relation to a specific object and which are supposed to be displayed whenever the object they are part of is displayed.

The representation of sub-assemblies is similar to part-whole relations, except that the arc "sub-assems" is used instead of "sub-parts".

m30(object d10 sub-assems d9 modality function) (22)

2. Clusters

Printed circuit boards sometimes show groups of objects that stand in a logical relation to each other, comparable to a part-whole relation. Nevertheless they are neither sub-parts nor sub-assemblies. Sub-parts and sub-assemblies have a *main object* that is itself displayable, i.e. that has a form. However, a grouping of components might consist of objects of the same size and importance, none of which deserves the status of main object. Fig. 4 shows a voltage divider and a T filter which are typical examples of such circuits.

A grouping which exists only as an abstraction is called a *cluster*. If one combines the concept of cluster with the concept of level a dilemma emerges. Either the abstract object is left out of the hierarchy (which is undesirable, because anything that seems natural to a person should be directly representable in the network (LPH)), or the abstract object is put in the hierarchy and the objects of the cluster are made its parts. But now the idea of creating simplified displays by limiting the number of levels displayed does not work any more, because the abstract object is not displayable in the same sense as real objects are. Moreover if one is willing to give an abstract object a symbolic form, then both the symbolic form as well as the cluster elements would be displayed if one wants to see all the levels of the part hierarchy. This would complicate the display unnecessary.

Our answer to this problem is to create an additional hierarchy which stands somewhere in between a part hierarchy and an abstraction hierarchy. If A is an object (without form) which has sub-clusters B, C, and D then A will be displayed *only* by displaying B, C, and D. However if a partial display is enforced in a way that would exclude the level of B, C, and D from showing, A will be displayed symbolically by a box, akin to the display format in block diagrams. Fig. 5 shows the new display format for Fig. 4. The network representation of a sub-cluster is shown by (23).

m31(object d10 sub-clusters d9 modality function) (23)

E. Attributes and Attribute Mappings

One important factor in designing a system based on graphical deep knowledge is a clear separation between icons and the objects that are represented by these icons, an observation that has been made by others also [21]. This separation forces one to

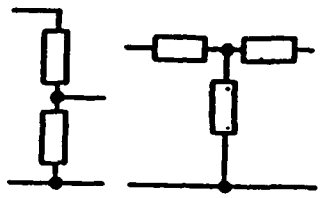


Figure 4

A Voltage Divider A "T"-Filter

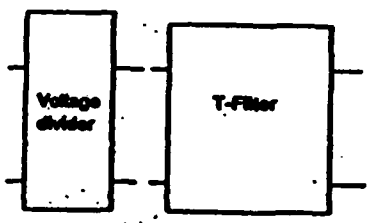


Figure 5

Abstract Pictorial representations for

Figure 4

distinguish between attributes of objects and attributes of pictures. A typical example of an attribute of a picture is *blinking*. On the other hand, *faultiness* is an attribute of a component, not of the picture of a component. Attributes like faultiness cannot be displayed directly and therefore have to be symbolized with pictorial attributes.

Attributes are represented by the name of an attribute-class with 0 to 3 positions for attribute values. The following examples show an attribute with no attribute-value (24), an attribute with one position containing the value "faulty" (25), and an attribute with two positions containing the values "left" and "90" (26). No need for any attributes with more than three positions for attribute-values has arisen yet.

```
m33( object d1                                              (24)
     attr m33( attr-cls new modality function))
m34( object d1                                              (25)
     attr m35( attr-cls state
               attr faulty modality function)
m36( object d2                                              (26)
     attr m37( attr-cls rotated
               attr1 left attr2 90 modality function)
```

The attribute statements (24) - (26) apply to objects as opposed to pictures.

Attribute classes are linked to functionals that can be applied to form functions. Such a functional is called a *modifier function*. (26) asserts that d2 has the attribute-class "rotated" with two values "left" and "90" (degrees), which requires a change to its form before it can be displayed correctly. Therefore a modifier function that rotates forms is bound to this attribute class and the form of d2 is passed to it as first argument. The attribute values (marked by the axes attr, attr1, attr2 and possibly attr3) are passed in correct order as additional arguments to the modifier function.

The binding of an attribute-class to a modifier function is asserted in the knowledge base. This makes it amenable to easy change by the user. Utterances like

```
"Represent the state faulty by red color and                (27a)
the state good by blue color."
```

have led to this representational decision (linearity principle). In (27b) the complete mapping necessary for (27a) is given.

```
m38( attr state                                             (27b)
     mod-func color
     modality function
     val1 m39( expressed faulty expressed-by red)
     val1 m40( expressed good expressed-by blue))
```

It binds the attribute-class "state" to the modifier-function "color". The two sub-structures at the end of the val1 arcs show value mappings between object attributes and picture attributes. The object attribute of faulty state is represented by the picture attribute of red color. val1 corresponds to attr1 and specifies value mappings that apply to the first attribute position.

Sometimes one encounters numerical attribute values as in (26). Representing a mapping between two large lists of numbers would be unwieldy to impossible. Luckily in many practical cases the relation between the attribute value of a picture and the attribute value of the corresponding object is an identity function. This case is taken care of by using the attribute value of the object if there is no explicit mapping from object value to picture value. This also can take care of a mapping that is the identity function except for a few singularities.

If a more complicated function is necessary to transform from object attribute-value to picture attribute-value then the mapping function has to be integrated in the attribute functional itself.

The method used to associate an actual function with an attribute-class is identical to the method used for form functions. The node specifying a modifier-function is at the same time the name of a LISP function.

An important finding of our work has been that there is inheritance along part hierarchies, something we have not yet read in the literature. For instance, if an object is represented with an attribute like "scaled", then one would want all its parts to inherit this attribute. Even more interesting is the fact that this inheritance does not apply to all attributes and depends on the attribute-class itself. If one assures, for instance, the faultiness of an object then it would defeat the whole purpose of a maintenance system to have all its parts inherit faultiness.

Given that one can easily express a fact like "scaling is inheritable", one should also (linearity principle) have a correspondingly simple representation in the network which is exactly what has been implemented.

```
m41( inheritable size)                                      (28)
```

If (28) is part of the current knowledge base then the attribute-class "size" will apply to the parts of all objects for which the size attribute has been asserted.

IV IMPLEMENTATIONAL STATE

TINA is going through its fourth cycle of implementation which is done in Franz LISP on top of SNePS. All of the shown knowledge structures (and more) are representable and retrievable from the network knowledge base, and most of them are interpreted in a way consistent with the descriptive semantics given in this paper. An older version of "TINA" has been applied to a real circuit board used for telecommunication purposes (PCM board). The IMD system described has been used for the Adder-Multiplier only.

V CONCLUSIONS

The problem of Intelligent Machine Drafting has been introduced, and it was argued that it is a theoretically interesting AI problem which is sufficiently different from other CAD techniques to deserve separate investigation. The class A"M" has been defined informally, and a few additional restrictions of the current IMD implementation for objects of this class have been given. The definition of Graphical Deep Knowledge and a (partial) task domain analysis of this area have been presented. A number of representational primitives have been introduced by way of example. These primitives comprise structures for representing knowledge about forms, concrete and fuzzy positions, and attributes. Positions have been differentiated into absolute and relative positions with explicit and implicit reference objects. Pixel based coordinates, body coordinates and reference-object coordinates have been introduced. Part hierarchies have been discriminated into real part hierarchies, sub-assemblies, and abstraction-hierarchy like clusters of objects. The derivation of some of these structures based on the "linearity principle" has been demonstrated, by presenting examples for motivating natural language utterances. A generator function which creates graphical representations from a knowledge base containing the indicated structures has been implemented.

References

1. D. C. Brown and B. Chandrasekaran, "Design Consideration for Picture Production in a Natural Language Graphics System," *Computer Graphics* 15(2) pp. 174-207 (July 1981).
2. Stuart C. Shapiro, Sargur N. Srihari, Ming-Ruey Taie, and James Geller, "VMES: A Network Based Versatile Maintenance Expert System," *Proc. of 1st International Conference on Applications of AI to Engineering Problems*, pp. 925-936 Springer Verlag, (April 1986).
3. Stuart C. Shapiro, "The SNePS Semantic Network Processing System," pp. 179-203 in *Associative Networks: The Representation and use of Knowledge by Computers*, ed. Nicholas V. Findler, Academic Press, New York (1979).
4. Stuart C. Shapiro, "Using Non-Standard Connectives and Quantifiers for Representing Deduction Rules in a Semantic Network," *Presented in Tokyo at: Current Aspects of AI Research*, (Aug. 27-28, 1979).
5. Stuart C. Shapiro and James Geller, "Artificial Intelligence and Automated Design," *1986 SUNY Buffalo Symposium on CAD: The Computability of Design*, SUNY at Buffalo, (Dec 6-7, 1986).
6. Stuart C. Shapiro and William J. Rapaport, "SNePS Considered as a Fully Intensional Propositional Semantic Network," *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp. 278-283 (1986).
7. Hajime Mori, Fujio Tanaguki, Masahiro Annaka, Satoshi Goto, and Tatsuo Ohtsuki, "Advanced Interactive Layout Design System for Printed Wiring Boards," pp. 495-523 in *Hardware and Software Concepts in VLSI*, ed. Guy Rabat, Van Nostrand Reinhold, New York (1983).
8. Isao Shirakawa, "Routing High Density Printed Wiring Boards," pp. 452-479 in *Hardware and Software Concepts in VLSI*, ed. Guy Rabat, Van Nostrand Reinhold, New York (1983).
9. Helmut Jansen, Erhard Nullmeier, and Karl-Heinz Rodiger, "Handsketching as a Human Factor Aspect in Graphical Interaction," *Computers and Graphics* 9(3) pp. 195-210 (1985).
10. M. C. Maguire, "A Review of Human Factors Guidelines and Techniques for the Design of Graphical Human-Computer Interfaces," *Computers and Graphics* 9(3) pp. 221-235 (1985).
11. Michael Hausman and Peter Scheie, "The Design of SWYSS, a Dialgue System for Scene Analysis," pp. 143-201 in *Natural Language Communication with Pictorial Information Systems*, ed. Leonard Bole, (1984).
12. Frank Zydbel, Noton R. Greenfeld, Martin D. Yonke, and Jeff Gibbons, "An Information Representation System," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pp. 978-984 (1981).
13. Mark Friedell, "Automatic Synthesis of Graphical Object Descriptions," *Computer Graphics* 18(3) pp. 53-62 (1984).
14. Stephen M. Kosslyn and Steven P. Shwartz, "A Simulation of Visual Imagery," *Cognitive Science* 1 pp. 265-295 (1977).
15. Stephen Michael Kosslyn, *Image and Mind*, Harvard University Press, Cambridge MA (1980).
16. Zenon W. Pylyshyn, "The Imagery Debate: Analogue Media versus Tacit Knowledge," *Psychological Review* 88(1) pp. 16-45 (1981).
17. Zenon W. Pylyshyn, *Computation and Cognition*, MIT Press, Cambridge MA (1984).
18. Lofti A. Zadeh, "Commonsense Knowledge representation Based on Fuzzy Logic," *Computer*, pp. 61-65 (Oct. 83).
19. Stuart C. Shapiro and The SNePS Implementation Group, "SNePS User's Manual," SNeRG Bibliography #31, SUNY at Buffalo (Sept. 1983).
20. Mary Angela Papalaskaris and Lenhart Schubert, "Parts Inference: Closed and Semi-Closed Partitioning Graphs," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pp. 304-309 (1981).
21. Fanya S. Montalvo, "Diagram Understanding: The Intersection of Computer Vision and Graphics," AI Memo 873, MIT (Nov. 1985).

Appendix A6:

"Artificial Intelligence and Automated Design"

Shapiro, S.C.
Geller, J.

ARTIFICIAL INTELLIGENCE AND AUTOMATED DESIGN

Stuart C. Shapiro
James Geller

Department of Computer Science
State University of New York at Buffalo
Buffalo, New York

Artificial Intelligence (AI) offers to the design task the use of powerful systems that can be knowledgeable assistants to the human designer. Knowledge Representation techniques can be used to specify the ontology and epistemology of the particular design task so an Intelligent Interface, in general, and an Intelligent Drafting assistant, in particular, can discuss the task with the designer using the same concepts that he uses. Investigating Knowledge Representation formalisms for such aids in the context of developing a Versatile Maintenance Expert System (VMES) has uncovered a number of interesting concepts that seem useful for a wider class of design domains. These concepts are presented after a general discussion of the role of AI in design, and an introduction to a particular AI Knowledge Representation system. The role of design aids and Intelligent Interfaces in VMES is presented as an example of the use of such systems.*

*This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB DC 20332 under Contract No. F30602-85-C-0008, which supports the Northeast Artificial Intelligence Consortium (NAIC).

☐ ARTIFICIAL INTELLIGENCE AND DESIGN

The task of design presents intelligent humans with a large number of complicated problems. Artificial intelligence (AI) is the research area which attempts to discover how to program computers to solve the sort of problems intelligent humans tackle. One use of AI in design might be to have an AI system that would do the design itself, perhaps viewing design as a search through a design-problem space. In this paper, however, we will discuss two aspects of the application of AI as design aids for human designers — the application of Knowledge Representation to drafting systems, and the use of Intelligent Interfaces. After some introductory remarks, we will give a brief introduction to the AI system we are using, present some results of our investigations into the applications of AI to design, and, finally, show how this fits into a Maintenance Expert System we are developing.

The Role of Knowledge Representation

Modern computerized drafting systems supply their users with a wealth of powerful modeling tools. A typical drafting system deals with objects, their visual and non-visual attributes, and their mappings into graphical representations. However, such a system is only a powerful set of pens, it is not an assistant that "knows" what the designer is talking about. To be intelligent, an assistant must be knowledgeable. Knowledgeable computer systems are known as "Knowledge-Based Systems" (KBSs), and are a very active area of AI research and development.

We can identify three roles that people play in the design and use of KBSs. First, there are people who design and implement KBSs without regard to any particular application domain. We can refer to such people as the KBS Designers, and to the results of their efforts, using terminology from the field of Expert Systems (ESs), as "KBS shells." Second, there are those who particularize KBS shells to given application domains. They are called "Knowledge Engineers" (KEs) in the ES world, and we can refer to the results of their efforts as KBSs *simpliciter*. Finally, there are the "end-users" who use KBSs as tools to get particular jobs done.

The job of a KE is usually perceived to be interviewing a person already knowledgeable (at an expert level) in the application domain, and recording that person's knowledge in a form that the KBS shell can use. However, if the KBS shell is flexible enough, there is an additional task for the KE: to design the "form" in which the knowledge is to be recorded. This task is the Knowledge Representation (KR) task, and we will refer to the KE performing this task as the "Knowledge Representation Engineer" (KRE). (The KRE's task has jocularly been called "notational engineering.") The KRE's first task is an analysis of the knowledge primitives in the domain. He must define the domain's ontology (the kinds of objects and attributes contained in the domain), and its epistemology (the sorts of things one may

know about the domain, and the ways of knowing them). A flexible KBS shell will permit the KRE to do this declaratively, *i.e.* without reprogramming the shell.

The KRE can supply a vocabulary of conceptual objects, relations, and attributes without a limit on the level of object abstraction. For example, one can take the system's representation of an object, and its representation of the depiction of the object on the screen, and create an explicit nonprocedural mapping between them. This mapping itself can be reified, which makes it in turn amenable to serving as an object in a propositional context. This example only involves two levels of abstraction and is frequently useful. For instance, it may be used to assert the validity of a mapping that might be limited to particular circumstances.

The declarative representation of these objects has the additional benefit of placing them in the domain of possible end-user queries. Whatever is a concept for both system and user can be discussed by them. The user can tell the system about them, and can ask the system what it currently knows about them. The system can have rules that specify how to reason about them, how to derive new attributes from old ones, and even under what circumstances to infer the existence of objects it hasn't been explicitly informed about.

A Knowledge-Based drafting system can be an intelligent assistant to a designer, rather than just a powerful drawing tool.

Intelligent Interfaces

Recently, there has been increased interest in the contributions AI can make to the design of interfaces. There was both a workshop and a panel on Intelligent Interfaces at the 1986 AAAI sponsored National Conference on Artificial Intelligence, and DARPA has recently funded a program on Multi-media Interfaces.

Our own view, [Shapiro 1986a] is that an intelligent interface needs the following capabilities: it should know about the topic under discussion, not merely be an isolated, modular, general purpose interface; it should know about communication issues, including what is on the screen, and the relationship between what is being communicated and the way it is being communicated; it should have a user model, so it has an idea of what the user knows, doesn't know, and what the user is trying to accomplish. The KBS-based drafter we are developing can be seen as an appropriate intelligent interface to a more extensive design system.

General Introduction to SNePS

The SNePS Semantic Network Processing System [Shapiro 1979; Shapiro 1986b] is the KBS shell we use, and we will use the SNePS formalism in the

remainder of this paper. For the reader not familiar with SNePS, we will first give a short introduction to the basic properties that distinguish it from other semantic network systems.

SNePS, unlike semantic network systems of the KL-ONE, KRYPTON family, [Brachman 1985; Brachman 1983] but like Anderson and Bower's HAM, [Anderson 1973] is a propositional semantic network system. *i.e.*, the main ingredient of SNePS networks are assertions, constructed from case grammar-like frames [Fillmore 1968]. This does not imply that SNePS cannot support KL-ONE type class hierarchies and inheritance [Tranchell 1982], but that this feature is less prominent in SNePS. SNePS is a fully intensional knowledge representation system [Shapiro 1986b] — it can represent imaginary, non-existing, and even impossible objects, as well as abstract objects, and multiple guises of a single object as if they were separate objects.

SNePS handles full predicate logic with universal, existential, and numeric quantification. A number of non-standard connectives that improve expressibility are available, including both a default operator and a true negation. SNePS supports forward, backward, and bidirectional inference, in contrast to many other systems which permit reasoning in only one direction. For instance, the OPS5 expert system shell does only forward inference, whereas PROLOG does only backward inference. In SNePS, the same rule syntax can be used for either type of reasoning; there are no specific forward or backward rules. SNePS permits the use of recursive rules, either directly recursive or indirectly recursive [McKay 1981]. A relevance logic based [Anderson 1975; Shapiro 1976] extension to SNePS permits its use as a truth maintenance system [Martins 1983].

Another advantage of SNePS is the total order independence of rules and clauses in the rules, in effect eliminating the painful mixed procedural-declarative semantics of PROLOG. This higher degree of flexibility permits very natural representations, especially for natural language rule expressions. However, the required computation times are usually longer than for PROLOG programs.

Although the major purpose of SNePS is not to be a functional model of the brain, as opposed to, for instance, Anderson's ACT system [Anderson 1983], SNePS has been designed with a high degree of cognitive validity in mind. This is expressed by a differentiation between conceptual and non-conceptual relations, by the impossibility of PROLOGish retract-like forced forgetting (except for debugging purposes), and by the accessibility of all information about a concept from the concept itself.

A number of different SNePS interfaces have been designed, containing several natural language parser/generators for subsets of English, a frame-like editor, a logic programming language, and several graphics interfaces. In our description of knowledge structures we will liberally use the "Lispish" notation of the SNePS User Language (SNePSUL), or our standard graphical representation of SNePS networks.

Knowledge Representation in SNePS

In this section, we will discuss an example SNePS network to introduce the syntax and semantics of some of the representational structures we use in our work on VMES, the Versatile Maintenance Expert System [Shapiro 1986c]. Figure 9.1 shows an Adder-Multiplier, a simple experimental device that has been used in the field of hardware maintenance research by a number of people. This object consists of three multipliers and two adders. Figure 9.2 shows part of the semantic network that describes this device. Rectangles in Figure 9.2 represent concepts of real or imaginary objects. Circles represent propositions about these objects. The network can be read as follows: D1 is an object of type M3A2; D1A1 is of type Adder and is a part of D1; D1M1 is of type Multiplier and is a part of D1; D1A1F1 is a Full Adder and is part of D1A1; etc.

The SNePSUL commands that create the network of Figure 9.2 are:

```
(define part-of object type)

(build object  D1A1
       type    Adder
       part-of D1)

(build object  D1M1
       part-of D1
       type    Multiplier)

(build object  D1
       type    M3A2)
       part-of D1A1
       type    Full Adder)

(build object D1A1F1
       part-of D1A1
       type    Full Adder)

(build object  D1A1F2
       part-of D1A1
       type    Full Adder)
```

The first define command defines the arcs to be used in the system. Arcs can be followed, for retrieval purposes, in either the forward or backward direction, guaranteeing the universal accessibility of every node from every other node that is related to it.

The set of build commands creates the actual network. Note that every build command will result in the creation of one "m..." node. These

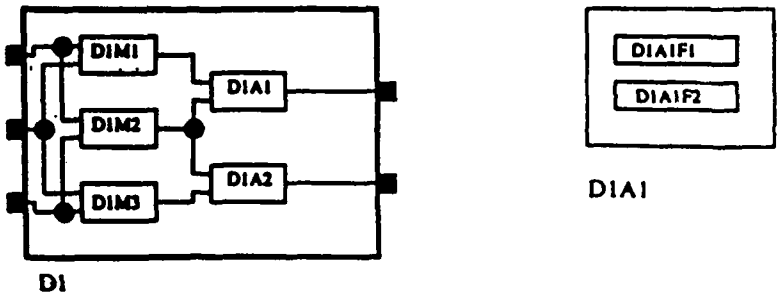


Figure 9.1. The Adder Multiplier and one of its parts.

nodes, as described earlier, correspond to propositions in the system and cannot be created directly by the user. In other words, it is not possible for the user to create an arc connecting two nodes named by him, guarding users against creating non-conceptual propositions, objects the SNePS theory of mind does not permit.

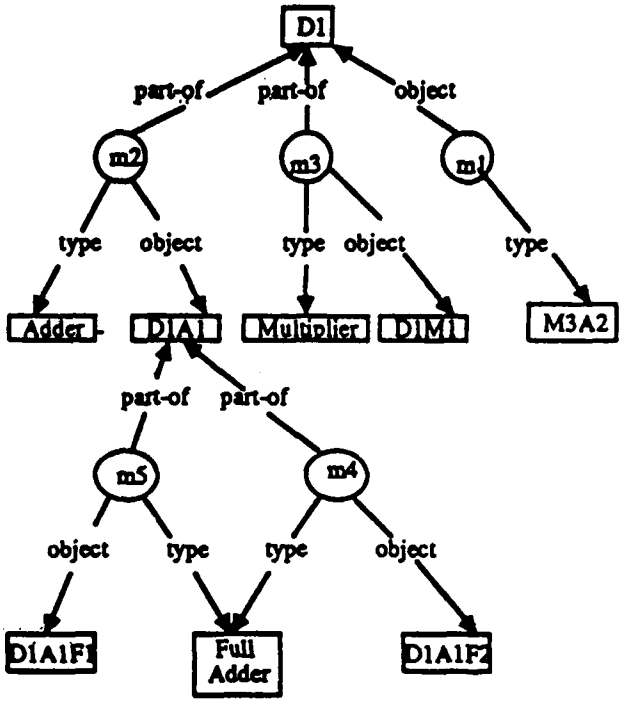


Figure 9.2. A piece of semantic net.

□ AN ANALYSIS OF IMPORTANT ELEMENTS OF DESIGN KNOWLEDGE

Having introduced the SNePS KBS shell, we will now discuss the ontology and representational constructs that we, in our role as KREs, have found to be necessary for creating descriptions of graphical depictions of simple circuit boards.

Objects and Forms

The first fundamental unit we need to deal with is the displayable *object*. In order to create a picture of an object it is necessary to specify a *form* for it. Every form has a dual role. On the one hand, it can be used to create a picture of that form. On the other hand, it is a conceptual unit in the knowledge representation system and can be manipulated as such. Picture creation is done by a Lisp graphics function whose name is identical to the form concept in the network, and whose arguments are the coordinate positions of the place the form is to be drawn. So, if the form of a particular gate is specified by the function gate-form, the gate would be drawn at position (100, 300) by evaluating the Lisp form:

```
(gate-form 100 300)
```

The degree of specificity of a form varies. While the form of an integrated circuit or a transistor is totally fixed, the form of a wire is dependent on the position of the ports it connects. If a user wishes to display an abstract object then he has to supply a symbolic form for it.

Positions

The next essential ingredient for a drafting system is the concept of *positions*. There are several possible ways of specifying positions. In a traditional CAD system, positions are only expressed in an absolute or relative manner based on coordinate values. This is an ability that a KBS should also have. However, knowledge-based design systems should also be able to deal with relational specifications, such as the specification that a certain element should be near or to the left of another element. This, of course, introduces a certain fuzzyness in the representation. However, in many cases this is exactly what a designer would like. It permits him to think in concepts that are natural to him, and it avoids unnecessary specificity. In other words, a knowledge-based drafting system permits one to specify spatial relations with a reasonable degree of imprecision.

The following SNePSUL commands show first our representation for relative coordinate positions, and then for fuzzy positions:

```
(build object   gate-1
       relpos   (build x 100 y 200)
       rel-to   gate-5
       modality function)

(build object   gate-2
       relpos   left
       rel-to   gate-1
       modality function)
```

The first SNePSUL command will create a piece of SNePS network representing the proposition that gate-1 is 100 units to the right and 200 units above gate-5. The second one asserts that gate-1 is to the left of gate-2. The `modality` slot is used to differentiate between different arrangements of an object in a functional representation (wire plan) and a physical representation (picture of the board).

Attributes

Attributes can either be of objects or of pictures of objects. An example of an attribute of a picture is *blinking*. A blinking picture can help a user focus his attention on a currently interesting object, without expressing anything about the object itself.

An example object attribute we have been using is the faultiness of a gate. The proposition that gate-1 is faulty would be represented by the network built by the command:

```
(build object   gate-1
       attr     (build atrb-cls state
                       atrb     faulty
                       modality function))
```

In order that the system know how to display a faulty gate, we tell it that the state attribute maps to the `state-to-color` function:

```
(build attr     state
       mod-func state-to-color)
```

Each attribute function, such as `state-to-color`, is actually a functional that takes a form function and an attribute value as arguments, and returns a modified form function. So, again, if gate-1 had the form represented by the function *gate-form*, and given that gate-1 is in the state of being faulty, and that the attribute function for state is *state-to-color*, gate-1 would be

displayed as faulty at coordinate position (100, 300) by evaluating the Lisp form:

```
(funcall (state-to-color #'gate-form 'faulty)
         100 300)
```

Notice that representing different attribute dimensions (state, color, size, etc.) by different attribute functionals explicates the way that different attribute dimensions are, in fact, different.

In this technique, the information of how to display a faulty gate is procedurally encoded in the state-to-color functional. An alternative is to store the information declaratively in the network, such as by a proposition built by the command:

```
(build attr     state
       strb     faulty
       mod-val  red)
```

This proposition says that the attribute of being in a faulty state is to be shown by making the display red. The fact that red is a value of the color attribute is stored by a separate proposition.

Class Hierarchy

An important feature of most knowledge representation systems is their ability to handle *classes* of objects (and also hierarchies with many levels of classes). This permits a user to associate an attribute with an entire class instead of a single object. For example, one could express the fact that all integrated circuits expect ground potential on their pin 0 by associating this fact with the class of all integrated circuits.

Classes have two important features that are valuable for design systems (and KR systems in general). The first is that by asserting that an object belongs to a certain class, a lot of new knowledge is immediately available about it. This is called inheritance along a class hierarchy. The other valuable feature is that this type of representation seems to correspond to the way people organize their knowledge. Therefore the naturalness of the use of classes also improves the general communication between user and system.

Part Hierarchy

Another feature that is common in AI systems is the use of *part hierarchies*. Much of the knowledge about physical objects can be organized as facts that express a part-whole relation between different objects. This applies also and especially to design systems.

Our own research has shown that the concept of inheritability which was mentioned for class hierarchies is also applicable to part hierarchies, but with a difference that we have not seen discussed in the previous literature. For instance, the attribute of a special transistor of being "twice as large as an average transistor" is inheritable by its parts. On the other hand, if a circuit board is known to be faulty, nobody would want this attribute to be inherited by all its parts. That would defeat the very purpose of a diagnosis system.

In class hierarchies, the only attributes that are not inheritable, are those that apply only to classes. For example, the cardinality of a class is not applicable to, let alone inherited by, its individual members. In the part hierarchy, however, there are non-inheritable attributes, such as faultiness, that are applicable to sub-parts, just not to be inherited by all of them.

The representation that we are using for inheritable attributes is the same as the representation for non-inheritable attributes, and is, in fact, identical to the example of faultiness given in an earlier section. However it is possible to assert in the network about a certain attribute that it is inheritable, simply by pointing to it with an `inheritable` arc. For example:

```
(build inheritable size)
```

The display program which interprets the network automatically queries for inheritability if it has to expand an object with attributes into parts. The results of this query determine whether or not the parts of the object are displayed as having the attribute.

Inheritability, as an attribute of other attributes, is a meta-attribute. The fact that we are representing it explicitly and declaratively gives the user the power to experiment with different attributes, and to postpone the decision about which of them is inheritable.

Our findings about inheritance can be extended to other hierarchies, which we refer to as *relevance hierarchies*. Relevance hierarchies are an abstraction of a number of different hierarchies used in the literature, including topic hierarchies [Haan 1986] and hierarchies of spatial universes (containment hierarchies) [Fahlmann 1979].

□ THE VMES SYSTEM

The research described in this paper is a part of the VMES (Versatile Maintenance Expert System) project, which deals with hardware maintenance for mixed analog and digital circuit boards. By using the features of a knowledge based architecture, a high degree of versatility has been achieved [Shapiro 1986c].

The specific significance of our work is that frequently electronic devices have fairly short life cycles. A new board is designed and quickly comes

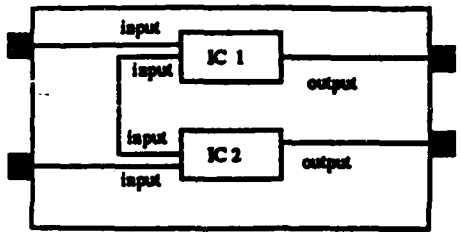


Figure 9.3. An invalid connection.

into use in the field. There is little time to design elaborate test procedures or equipment, or to educate a large number of technicians and users. Usually, the only real expert on the device is its designer, and he is already involved with another project when the first problems in the field come up. Our research is directed toward the design of a KBS-based drafter that the designer will use to help design a new device. This design stage will be the "Knowledge Acquisition" stage of the VMES, which will then be able to advise maintenance technicians on the maintenance and repair of the device that it helped design.

The maintenance system can also be used as a part of the design system, since it can be used to detect impossible designs which do not conform to certain integrity constraints. An example of such an impossible design in the circuit board domain would be if a new device that is described to the system has two chips with their input ports connected to each other, but neither connected to an outport of any other chip (Figure 9.3). Another example would be if two points are electrically connected to each other by two separate wires (Figure 9.4).

VMES implements a large number of the concepts which have been described in the previous sections, i.e. part and class hierarchies, inheritance, attributes, etc. It expects to talk to two different types of end-users. On the one hand are maintenance technicians with a limited amount of education and training. On the other hand are the designers that enter a description of a new device into the system. These two types of end-users

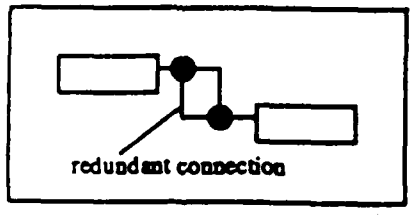


Figure 9.4. A redundant connection.

have different user interfaces, but both interfaces are required to be natural and user-friendly.

The need to create descriptions of circuit boards quickly and without "programming" requires a system that has fairly general knowledge about circuit boards, and that can be adapted to a new device in a short time and with a natural dialogue. To achieve this the system has to *understand* much about the objects of the domain, like wires, inverters or integrated circuits. The use of a Knowledge Representation language is a precondition for achieving such understanding. Use of a component library also permits a rapid change from one device to another. If a new device does not contain any new components, then it is only necessary to describe the new wiring.

Our approach to the design of Intelligent Interfaces may be explained by a description of three interfaces that are part of VMES. The main user interface is a Knowledge-Based graphics component. This program, named `display`, takes a piece of semantic network as argument and uses it to generate a pictorial representation of the stored knowledge. `display` works as a generator, quite comparable with a natural language generator. Only redundant permanent auxiliary storage is used by `display`. In other words, the semantic network plus the Lisp functions describing primitive forms are the only knowledge sources for the computation and creation of device depictions.

We are working on displaying devices under the assumption that no coordinate positions are given. We refer to this activity as intelligent machine drafting (IMD). We are attempting to provide a procedural model of some of the knowledge that a draftsman has about space and arrangement of electronic components. `display` tries to arrange components of the system in what it "thinks" is a graphically appealing way, using several variations of an equal-spacing algorithm. Unlike VLSI routing or layout programs, which usually try to find some space-optimal solution, `display` assumes that there is ample space to solve the placing problem.

The second interface is a natural language understander (NLU), implemented by using an augmented transition network (ATN) [Woods 1970; Shapiro 1982] semantic grammar. A user can create classes of objects, assign (predefined) forms to them, name members of these classes, assign them attributes, and then display them, all with commands from a (fairly limited) subset of natural language. The NLU uses the same KR constructs as are used by `display`. This enables it to demonstrate its understanding of declarative sentences by drawing the object(s) mentioned using appropriate graphic indicators of the asserted attributes.

The third interface is the `readform` facility, which allows a user to create Lisp form functions simply by drawing objects. `readform` permits a user to create pictures of objects from simple primitives like lines, circles, boxes etc. He can also design a form off to the side, on a kind of scratch pad, and then add this form repeatedly to the object being designed. `readform` will assume that the form created on the side is the form of a class of

objects, and that the repeatedly added instances are members of that class. These members will also be assumed to be parts of a main object, consisting of the primitives placed before and after using the scratch pad. readform verifies some of its assumptions by querying the user, e.g. asking for a name of the suspected class. If the user supplies the requested names then readform will create a network structure that asserts the class and part relations and will even store the positions of the parts relative to their super object.

□ CONCLUSIONS

AI offers to the design task the use of powerful Knowledge-Based System shells. Knowledge Representation Engineers particularize these KBS shells to the particular design domain by specifying the ontology and epistemology of the domain. This permits the end-users to discuss the design task with the KBS as if it were a knowledgeable assistant.

We discussed two aspects of KBSs useful for design. Intelligent Interfaces know the task being performed, know about the objects, relations, and attributes being discussed, and know how to express these concepts to the user. Intelligent Machine Drafters (IMDs) are knowledgeable assistants to the designer, besides being powerful drafting tools.

We have been developing a Versatile Maintenance Expert System (VMES) that would be able to help a maintenance technician repair a device that had been designed so recently that there would not have been time to give the technician training on how to repair it. The VMES would acquire its own knowledge of the device by serving as an IMD to the original designer.

In our roles as KREs for VMES, we have identified the following concepts as useful for an IMD and for an Intelligent Interface to a design system: objects; forms of objects; absolute, relative and "fuzzy" positions; attributes of objects and of pictures of objects; attribute functionals; object attribute to picture attribute mappings; class, part, and relevance hierarchies; and meta-attributes, such as inheritability.

□ ACKNOWLEDGMENTS

We would like to thank the other members of the VMES team, Mingruey R. Taie, Sargur N. Srihari, and Scott S. Campbell for valuable discussions; Dale Richards from RADC for administrative support; Bill Eggers, Michael Rosenzweig, Jim Carney, and Carl Mercer for working on several generations of "Readform"; and finally Lynda Spahr, our secretary, for being a pearl in general.

☐ REFERENCES

AAAI *Proceedings of the Fifth National Conference on Artificial Intelligence*, Morgan Kaufmann Publishers, Los Altos, CA, 1986.

Abbott C. "Introduction to the Special Issue on Computer Music," *ACM*, Computing Surveys, 17(2):147-289, June 1985.

Anderson A. and N. Belnap *Entailment: The Logic of Relevance and Necessity*, Princeton University Press, vol.1, 1975.

Anderson J.R. "A Spreading Activation Theory of Memory," *Journal of Verbal Learning and Verbal Behavior*, 22:261-295, 1983.

Anderson J.R. and G.H. Bower *Human Associative Memory*, V. H. Winston and Sons, Washington, D.C., 1973.

Ballard D.H. and C.M. Brown *Computer Vision*, Prentice Hall, 1982.

Brachman R.J., R.E. Fikes and H.J. Levesque "KRYPTON: A Functional Approach to Knowledge Representation," IEEE Computer, 16(10):67-73, 1983.

Brachman R.J. and H.J. Levesque *Readings in Knowledge Representation*, Morgan Kaufmann Publishers, Los Altos, CA, 1985.

Boden M.A. *Artificial Intelligence: How Machines Think*, Simon & Schuster, NY, 1985.

Brachman R.J. and J. Schmolze "An Overview of the KL-ONE Knowledge Representation System," *Cognitive Science*, 9(2):171-216, 1985.

Charniak E. and D. McDermott *Introduction to Artificial Intelligence*, Addison Wesley, Reading, MA, 1985.

Fahlmann S.E. *NETL: A System for Representing and Using Real-World Knowledge*, MIT Press, Cambridge, MA, 1979.

Fillmore C.J. "The Case for Case," *Universals in Linguistic Theory*, ed. E. Bach and R. T. Harms, Holt, Rinehart, and Winston, NY, pp. 1-88, 1968.

Gardner H. *The Mind's New Science: A History of the Cognitive Revolution*, Basic Books, NY, 1985.

Haan J. de and L.K. Schubert "Inference in a Topically Organized Semantic Net," *Proceedings of the Fifth National Conference on Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, pp. 334-338, 1986.

Hayes-Roth F.; D.A. Waterman and D.B. Lenat *Building Expert Systems*, Addison-Wesley, Reading, MA, 1983.

Hofstadter D.R. and D.C. Dennett *The Mind's I*, Bantam Books, NY, 1981.

Hunt M. *The Universe Within*, Simon & Schuster, NY, 1982.

IJCAI *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann Publishers, Los Altos, CA, 1985.

McCalla G. and N. Cercone "Approaches to Knowledge Representation," *Computer*, pp. 12-18, Oct. 1983.

McKay D.P. and S.C. Shapiro "Using Active Connection Graphs for Reasoning with Recursive Rules," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, pp. 368-374, 1981.

Martins J.P. *Reasoning in Multiple Belief Systems*, 203, SUNY at Buffalo, Dept. of Comp. Sci., 1983.

Nilson N.J. *Principles of Artificial Intelligence*, Tioga, Palo Alto, CA, 1980.

Peat F.D. *Artificial Intelligence: How Machines Think*, Simon & Schuster, New York, 1985.

Shapiro S.C. and M. Wand *The Relevance of Relevance*, 46, Indiana University, 1976.

Shapiro S.C. "The SNePS Semantic Network Processing System," *Associative Networks: The Representation and use of Knowledge by Computers*, ed. by Nicholas V. Findler, Academic Press, NY, pp. 179-203, 1979.

Shapiro S.C. "Generalized augmented transition network grammars for generation from semantic networks" *The American Journal of Computational Linguistics*, 8(1):12-25, 1982.

Shapiro S.C. and J. Geller "Knowledge Based Interfaces," *AAAI-86 Workshop on Intelligence in Interfaces*, ed. Bob Neches and Tom Kaczmarek pp. 31-36, August, 1986a.

Shapiro S.C., S.N. Srihari, M.R. Taie and J. Geller "VMES: A Network Based Versatile Maintenance Expert System," *Applications of AI to Engineering Problems*, 1986c.

Shapiro S.C. and W.J. Rapaport "SNePS Considered as a Fully Intentional Propositional Semantic Network," *Proceedings of the Fifth National Conference on Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, pp. 278-283, 1986b.

Tranchell L.M. *A SNePS Implementation of KL-ONE*, TR-198, Dept. of Comp. Sci., SUNY at Buffalo, 1982.

Winston P.A. *Artificial Intelligence*, Addison-Wesley, Reading, MA, 1984.

Woods W.A. "Transition Network Grammars for Natural Language Analysis," *Communications of the ACM*, 10:591-606, 1970.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.